Learning in Dynamic Data-Streams with a Scarcity of Labels

by

Conor Fahy

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

School of Computer Science and Informatics

De Montfort University

2019

Declaration of Authorship

The content of this submission was undertaken in the School of Computer Science and Informatics, De Montfort University, and supervised by Prof. Shengxiang Yang and Dr. Mario Gongora during the period of registration. I hereby declare that the materials of this submission have not previously been published for a degree or diploma at any other university or institute. All the materials submitted for assessment are from my own research, except the reference work in any format by other authors, which are properly acknowledged in the content. Part of the research work presented in this submission has been published or has been submitted for publication in the following papers:

C. Fahy and S. Yang. "Dynamic Stream Clustering Using Ants" *Advances in Computational Intelligence Systems* (pp. 495-508). Springer, Cham. 2017

C. Fahy, S. Yang, and M. Gongora. "Finding multi-density clusters in non-stationary data streams using an ant colony with adaptive parameters." *Proc. 2017 IEEE Congress on Evolutionary Computation* pp. 673-680, 2017.

C. Fahy, S. Yang, S. and M. Gongora. "C. Fahy, S. Yang, and M. Gongora. *"Ant colony stream clustering: A fast density clustering algorithm for dynamic data streams"*. IEEE Transactions on Cybernetics, 49(6): 2215-2228, June 2019 (DOI: 10.1109/TCYB.2018.2822552)".

C. Fahy and S. Yang. "Finding and Tracking Multi-Density Clusters in Data Streams", *IEEE Trans. on Big Data*, accepted, May, 2019.

C. Fahy and S. Yang. "Dynamic Feature-Selection for Clustering High Dimensional Data-Streams", *IEEE Access*, submitted June, 2019.

C.Fahy and S. Yang. "Classification in Dynamic Data Streams with a Scarcity of Labels" *IEEE Trans. on Pattern Matching and Machine Intelligence*, submitted April, 2019.

Abstract

Analysing data in real-time is a natural and necessary progression from traditional data mining. However, real-time analysis presents additional challenges to batch-analysis; along with strict time and memory constraints, change is a major consideration. In a dynamic stream there is an assumption that the underlying process generating the stream is non-stationary and that concepts within the stream will drift and change over time. Adopting a false assumption that a stream is stationary will result in non-adaptive models degrading and eventually becoming obsolete.

The challenge of recognising and reacting to change in a stream is compounded by the *scarcity of labels* problem. This refers to the very realistic situation in which the true class label of an incoming point is not immediately available (or will never be available) or in situations where manually labelling incoming points is prohibitively expensive. The goal of this thesis is to evaluate unsupervised learning as the basis for online classification in dynamic data-streams with a scarcity of labels.

To realise this goal, a novel stream clustering algorithm based on the collective behaviour of ants (Ant Colony Stream Clustering (ACSC)) is proposed. This algorithm is shown to be faster and more accurate than comparative, peer stream-clustering algorithms while requiring fewer sensitive parameters. The principles of ACSC are extended in a second stream-clustering algorithm named Multi-Density Stream Clustering (MDSC). This algorithm has adaptive parameters and crucially, can track clusters and monitor their dynamic behaviour over time. A novel technique called a Dynamic Feature Mask (DFM) is proposed to "sit on top" of these stream-clustering algorithms and can be used to observe and track change at the *feature* level in a data stream. This Feature Mask acts as an unsupervised feature selection method allowing high-dimensional streams to be clustered. Finally, data-stream clustering is evaluated as an approach to one-class classification and a novel framework (named COCEL: Clustering and One class Classification Ensemble Learning) for classification in dynamic streams with a scarcity of labels is described. The proposed framework can identify and react to change in a stream and hugely reduces the number of required labels (typically less than 0.05% of the entire stream).

Acknowledgments

I would like to thank my supervisors; Prof. Shengxiang Yang for the guidance, wisdom, and weekly meetings. Those meetings prevented this from ever getting too overwhelming. Dr. Mario Gongora for the support and advice during the PhD and also the preceding masters degree.

I'd like to thank everyone in the Centre for Computational Intelligence, especially all of the other PhDs, past and present, in GH4.56. Particular thanks to Dr. Jane Eaton and Dr. Muhanad Younis for the camaraderie, tea-breaks, and help with latex templates!

I would like to thank my parents John and Marie for pretty much everything. And my brother Declan, I probably would not be doing this without his advice and help over the years.

Finally, thanks to Jade for the love, encouragement, and support throughout. Especially in the final writing-up months when I was...difficult. Thanks for buying wine and pretending to be interested in artificial ants.

Cheers, all.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ACSC . . . . . Ant Colony Stream Clustering
AL . . . . . Active Learning
COCEL . . . . . Clustering and One Class Classification Ensemble Learning
E . . . . . Ensemble
F F1-Score
FS . . . . . Feature Selection
ILNS . . . . . Initially Labelled Non-Stationary Streaming
LS . . . . . Laplacian Score
MCFS . . . . . Multi-Cluster Feature Selection
mc . . . . . Micro Cluster
MC . . . . . Micro Classifier
MDSC . . . . . Multi Density Stream Clustering
MOA . . . . . Massive Online Analysis
MST . . . . . Minimum Spanning Tree
NN . . . . . Nearest Neighbour
OCC . . . . . One Class Classifier
P . . . . . Purity
PCA . . . . . Principal Component Analysis
PM . . . . . Pheremone Matrix
R . . . . . Rand Index
SoL . . . . . Scarcity of Labels
SSL . . . . . Semi-Supervised Learning
SVM . . . . . Support Vector Machine
SVDD . . . . . Support Vector Domain Description
VFDT . . . . . Very Fast Decision Tree

Chapter 1

Introduction

A data-stream is a continuously arriving sequence of data. Analysing this data in real time is a natural and necessary progression from classical data mining. Traditionally, a finite subset of this data would be collected, stored and then mined. However, if the underlying process is generating data in real time, it is preferable to analyse it in real time. By processing the data online we can make timely decisions, infer patterns and detect anomalies as they happen.

Data Streams pose new challenges for data mining and machine learning. Speed is crucial. The data needs to be analysed quickly and efficiently in order to prevent lag, bottle-necks and a potential loss of data. Under this constraint, typically only a single pass of the data is afforded. Available memory is another constraint. A stream is potentially unbounded but only a finite amount of memory is available. This is a consideration for both offline storage of processed data and also the memory available to an online algorithm. Consider the task of online classification, Bifet *et al.* outline four requirements which differ from traditional batch-learning [24]:

- **Requirement 1:** Process an example at a time, and inspect it only once
- **Requirement 2:** Use a limited amount of memory
- **Requirement 3:** Work in a limited amount of time
- **Requirement 4:** Be ready to predict at any time

In addition to these four, a fifth important requirement could be added:

- **Requirement 5:** Be able to effectively handle change

Change is one of the most challenging aspects when dealing with data-streams. In traditional batch methods, a model is trained on a finite subset of data assumed to be a full representation of the problem, decision boundaries are discovered and future

samples are assumed to be generated from the same distribution as the training sample. In a dynamic stream the characteristics of the data and decision boundaries change. This requires classifiers to be constantly updated. Simply recognising that change has occurred is a challenge, as is reacting and adapting to this change.

1.1 Change in a Data Stream

Let $S = [i^t]_{t=0}^{\infty}$ denote a stream where $i^t = (x^t, y^t)$, x is a vector in d dimensions which describes a class y at time t. Change in a stream can be sudden or gradual and these changes can be transient (in that the effect of the change disappears after a certain amount of time) or permanent. Change in a stream can occur on two different levels; the feature level and the concept level.

1.1.1 Change at Feature Level

Change at the feature level can occur in two ways; feature *drift* and feature *evolution*. Assuming an incoming instance x^t in d dimensions at time t, $x^t = \{{f_1}^t, \ldots, {f_d}^t\}$. Feature drift occurs if the importance, discriminatory power or relevance of a feature f_i changes over the course of a stream. For example, in text-mining the relevance of a particular word can change over time. In the presence of feature drift, after time δ, the relevance of a feature f_i changes, i.e. ${f_i}^t \neq {f_i}^{t+\delta}$

Feature evolution occurs when new features appear in the stream, for example, additional words might appear in a text stream and d, the dimensionality of x, changes. If feature evolution occurs, $d^t \neq d^{t+\delta}$

Sufficient change at the feature level will cause change at the concept level.

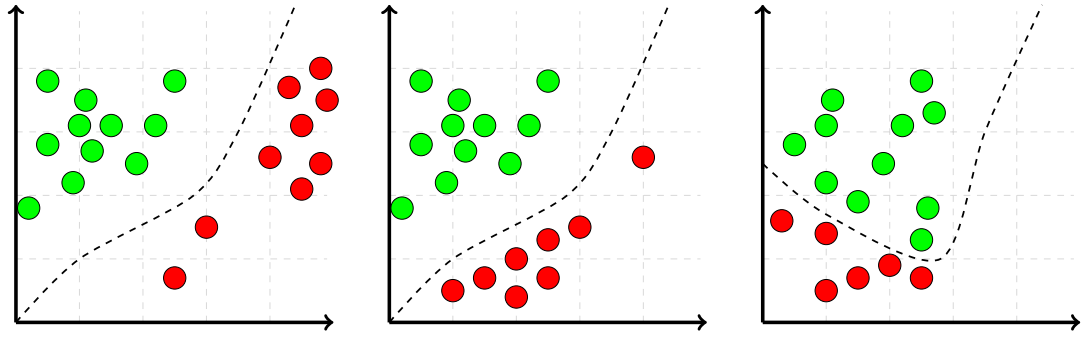
1.1.2 Change at Concept Level

Change at the concept level can also take the form of drift or evolution.

Assuming Y represents the set of all known classes; $Y = \{y_1, \ldots, y_n\}$, concept evolution occurs when an entirely new class y_k appears in the stream, i.e. $y_k \notin Y$. Obviously, samples from y_k should be recognised as a new class and the classification model updated accordingly. A real world example of concept evolution could take the form of network attacks. Over time, attackers develop new methods in response to increased security measures. These new classes of attack will be unfamiliar to existing models training on previously seen data.

Another change can occur in the form of concept drift. This occurs if the characteristics of the data change, or if the relationship between the data and the target

class changes. Given a vector x_i at time t describing concept y_i, represented as a conditional probability $P^t(y_i|x_i)$ (at time t vector x_i describes class y_i). In the presence of concept drift, after time δ, $P^t(y_i|x_i) \neq P^{t+\delta}(y_i|x_i)$.



(a) Original Data (b) Virtual Drift (c) Real Drift

Figure 1.1: Illustrative example of concept drift. (a) Original data, two classes with a decision boundary. Virtual drift is displayed in (b), this is a change in $P(x)$ but no resultant change in the decision boundary. Real drift is illustrated in (c), here there is a change in $P(y|x)$ and the decision boundary

Concept drift is usually described as either being virtual drift or real drift and the difference is illustrated in Figure 1.1. Virtual drift is a change in $P(x_i)$ without affecting $P(y_i|x_i)$, a change in the *conditional*. This change can be gradual, for example sampling from source distribution SD_i decreases and sampling from SD_j increases, or the change can be sudden, for example, the source distribution SD_i at time t is replaced by SD_j at $t+1$. It is usually assumed that virtual drift does not affect the decision boundary and has been described as simply an "incomplete data representation" [193] in the training data though, in practical terms, virtual drift can often lead to real drift. Real drift is a change in $P(y_i|x_i)$, for example at time t vector x_i describes class y_i ($P^t(y_i|x_i)$) but after time δ, x_i describes class y_j ($P^{t+\delta}(y_j|x_i)$). To illustrate the difference here we might consider an example of an industrial machine which generates senor-data in real time (pressure, flow, temperature, etc.). As the components degrade over time we would expect the sensor measurements to change, this is virtual drift as the machine is still functioning normally. If, however, the machine stops functioning normally but the sensor data remains the same we can say that real concept drift has occurred; the sensor data is the same as the previously functioning version, but the machine itself has has changed 'class' from functioning to non-functioning. Real drift can occur suddenly with no change in $P(x)$ or gradually, perhaps initially as virtual drift in $P(x)$. If real drift occurs without a change in $P(x)$ it is difficult to imagine how this could be recognised without access to the true class labels.

1.2 Scarcity of Labels

Assuming a tuple (x_i, y_i) where x_i is a data point with an associated class label y_i, the label y_i is the "ground truth" of x_i, *i.e.* it indicates what x_i *is* or represents. The scarcity of label problem refers to situations where data (x) is abundant but the associated labels (y) are scarce. This is a problem shared with traditional batch learning models, however it is exacerbated in a stream environment where unannotated data is continuously arriving.

In a strictly *supervised* environment, y_i, the true class label of x_i, is immediately available or easily accessible. This would often be the case in financial, or energy usage prediction. Let's say, for example, that we are classifying a stock as either increasing or decreasing in value, our classifier makes a prediction at t and the true label is available (and easily obtained) at $t + 1$. This labelled point is then used to update the classifier. In this example there is no scarcity of labels.

Dynamic classification with a scarcity of labels presents a more difficult challenge but is potentially of greater practical use. In these situations the true class label of an incoming point is not immediately available, a scenario referred as *verification latency* (for example in credit approval systems), or manually labelling incoming points by a human expert is labour intensive (for example in network security or social media posts). It might be trivial to label one instance but in high velocity streams it is just not realistic to label them all. In this case, some points can be labelled but this then raises a further question - *which* points should be labelled?

In a dynamic stream where change is expected, knowing *when* to update the model is a challenge. In periods of stability, there is no need to update the model or expend effort labelling incoming instances but when change begins to occur it needs to be recognised and the model updated. In a supervised setting this is easier as the model's classification accuracy will begin to degrade in a time of change and this degradation can be observed by comparing the predictions with the ground truth. This is difficult in situations where labels are scarce and impossible when labels are absent. Another realistic, practical scenario is where labelled training samples are initially available but the subsequent stream is unlabelled data drawn from a non-stationary distribution. Such a setting has been referred to as *initially labelled non-stationary streaming* (ILNS) [44]. In these cases a fully supervised approach is not possible and an *unsupervised* learning is perhaps a more suitable approach.

1.3 Learning Without Labels

In an unsupervised setting, data is available but none of the associated class labels are known, *i.e.* we have x but none of the corresponding y. Clustering is the most common form of unsupervised learning and has many definitions and descriptions. Broadly, it is the process of grouping a set of items in to sets so that items within the same set are more similar to each other than those in a different set. There are many clustering algorithms designed for the traditional batch setting but these are largely unsuitable for a streaming environment (for reasons covered in Section 2.1). Some of these algorithms have been extended to handle certain aspects of data-streams and many forms of stream-clustering algorithms exist.

The ideal stream clustering algorithm will adhere to time and memory constraints. It should require only a single pass of the data and summarise the data in a meaningful, interpretable way. In addition to the time and memory requirements, the clustering algorithm should be able to handle the various types of change. Discovered clusters should be online and available to accommodate incoming points. What these online clusters represent could be interpreted by a human expert and their behaviour monitored over time. As the stream progresses, data will age and become less relevant. So, some form of "forgetting" is required. In this way, as data is added and removed, the clusters could adapt to, and track, change in an organic way.

We could formalise this process using the same notation as in Section 1.1.2. Here C is set a of n discovered clusters; $C = \{c_i, \ldots, c_n\}$, and where previously $P(y_i|x_i)$ represented the probability of point x_i describing class y_i, $P(c_i|x_i)$ represents the probability of x_i being assigned to cluster c_i. If the meaning of c_i has been identified (perhaps with the summarised contents, or some representative samples), the potential relationship between an online cluster and a binary (one-class) classifier is apparent.

1.4 Motivation

Machine Learning applications have become ever more prevalent across a range of industries. These sophisticated machine learning techniques typically require huge amounts of annotated data. Such data sets typically need to be manually labelled in a time-consuming, labour intensive process. There is such a need for annotated data that an entire new sub-industry is emerging to accommodate it. Companies can be hired to do this [63], or the work can be crowd sourced [11].

These fully supervised methods are achieving ground-breaking results but the

process is long and could potentially be more efficient by developing methods which could remove (or reduce) the reliance on a fully labelled dataset. This is a familiar problem and is not new to the field. However, it is exacerbated in a streaming environment. In a high-velocity stream it is perhaps impossible to manually label every incoming point. Even if it were possible, in a non-stationary stream, data might already be irrelevant and out-dated by the time it has been manually annotated.

Classifying dynamic streams with a scarcity of labels requires some fundamental challenges to be addressed. How, without the ground truth, can we recognise change in a stream? How do we know which points are useful (and appropriate) for updating our classification model? Given these points, how can we use them to update our model? The work presented in this thesis is motivated by these questions.

1.5 Aims

The overall aim of this research is to investigate the potential of stream clustering as a foundation for online, dynamic classification. This overall aim can be deconstructed into two specific research goals:

- To develop a stream clustering algorithm which will act as the basis for subsequent classification methods. The algorithm should adhere to time and memory constraints, be able to handle the various types of change in a stream and allow for identified clusters to be tagged and monitored as the stream progresses.
- To develop and evaluate cooperative strategies for a stream clustering algorithm and an online classifier. Such a strategy should be able to recognise change and react accordingly.

1.6 Unique Contribution

- A novel stream clustering algorithm based on the sorting behaviour of ants in introduced. The algorithm; Ant Colony Stream Clustering (ACSC) [56] is shown to be faster and more accurate than comparative, peer stream-clustering algorithms while requiring fewer sensitive parameters.
- The principles of ACSC are extended in a second stream-clustering algorithm named Multi-Density Stream Clustering (MDSC) [58]. This algorithm has adaptive parameters and crucially, can track clusters and monitor their dynamic behaviour in a non-stationary stream.

- A novel technique called a Dynamic Feature Mask (DFM) [59] is proposed to "sit on top" of an existing stream-clustering algorithm and can be used to observe and track change at the *feature* level. This DFM also acts as an unsupervised feature selection method allowing high-dimensional streams to be clustered, which otherwise could not be.

- A novel framework for classification in dynamic streams with a scarcity of labels is proposed. COCEL: Clustering and One class Classification Ensemble Learning [60] can identify and react to change in a stream and hugely reduces the number of required labels (typically less than 0.05% of the entire stream).

1.7 Thesis Structure

The remainder of this thesis is organised as follows:

Chapter 2 presents a discussion of relevant, related work in the field of data-stream analysis.

Chapter 3 describes a novel method for clustering data-streams with artificial ants. The proposed method is motivated, described, and evaluated. In this chapter, the metrics and datasets that are used throughout the thesis are introduced and described.

Chapter 4 extends the clustering algorithm described in Chapter 3. It addresses the limitations of the original algorithm. Experimental and comparative results with the state-of-the-art are presented along with a case study using a data-stream of monitored atmospheric gases.

Chapter 5 presents a technique for finding and tracking change at the feature level. It is algorithm independent and is shown to improve clustering performance while speeding the underlying clustering algorithm. The technique is evaluated on four high dimensional data streams.

Chapter 6 outlines a framework for dynamic stream classification with a scarcity of labels. The framework incorporates the stream-clustering algorithm and an ensemble of OCCs. It is compared with the state-of-the-art across benchmark data and two data-streams which would be realistic applications of the proposed framework.

Finally, Chapter 8 concludes this thesis. A summary of the results and the main contributions are presented. Limitations are discussed. The wider application of the proposed methods along with potential extensions are considered.

Chapter 2

Related Work

This chapter presents an overview of relevant and related research. Established ideas are discussed along with recent advances in the field.

2.1 Clustering

Clustering, broadly, is the process of separating data into separate groups, so that data in one group are more similar to data in another group. The goal of clustering is to discover the intrinsic structure in a set of unlabelled data. Given some data-set $X = \{x_1, \ldots x_n\}$ where n is the number of unlabelled samples, the aim is determine a set of partitions $C = \{c_i \ldots c_k\}$, where k is the number of discovered partitions and each $c_i \in C$ describes a separate grouping or a 'cluster' within X.

A brief overview of classical clustering techniques is presented. Generally, a clustering algorithm falls into one of five families; partitional, hierarchical, grid, density, and model based. Each is discussed here, along with a sixth relevant type; Ant-inspired clustering techniques.

2.1.1 Partitional

Partitioning methods aim to split a dataset into a pre-determined (k) number of partitions. Each partition represents a cluster. After the partitions have been initialised (randomly or with some heuristic [159, 170]), the algorithm iteratively reassigns data from one partition to another until some objective function has been minimised. Example objective functions could be least-squares or least absolute difference. k-means [80] and k-medoids [97] are two of the most well-known partitional methods, along with CLARANS [142] and Partitioning Around Medoids (PAM) [98].

Partitional methods are relatively simple and intuitive, however they have a couple of characteristics which render them unsuitable for stream-clustering: only spherical (or hyper-spherical) clusters can be discovered. This creates a limitation on the type of concept they can discover and makes them prone to noise and outlier points. The second, more crucial drawback is that k must be specified. In a dynamic stream this number will likely change over time, especially in streams where concept evolution is expected.

2.1.2 Hierarchical

Hierarchical methods aim to group data into a hierarchical tree structure where each split represents a cluster. Strategies for this typically fall into two types; agglomerative (a bottom up) approach and divisive (a top down) approach. In the agglomerative approach, each point starts in its own cluster and pairs of clusters are merged as the algorithm moves up the hierarchy. Conversely, the divisive approach groups all data into a single cluster at the beginning and this cluster is split as the algorithm moves down the hierarchy. An example divisive algorithm is DIvisive ANAlysis (DIANA) [97]. Initially, all points are in a single cluster, then the largest cluster is split until each data point is separate. At each split, DIANA chooses the data point with the largest average dissimilarity and creates a new cluster. All points which are more similar to the new cluster rather than the original cluster are moved. Other popular hierarchical methods include CURE [75] and CHAMELEON [99].

A potential drawback for a strictly hierarchical method is that once a merge or split has been performed, it cannot be undone. This is not ideal in a dynamic environment and hierarchical methods are typically combined with other clustering techniques to cope with data streams.

2.1.3 Grid-Based

Grid-based techniques split the data space into a series of multi-dimensional discreet cells. Data is mapped to its corresponding grid cell and each cell stores summary information of its contents. Clusters are typically defined as neighbouring grids with sufficient data mapped to them or, depending on the granularity of each grid, one cell could itself constitute a cluster (if sufficient data is mapped to it). Popular grid-based algorithms are STING [190] and WaveCluster [173].

Grid-based methods have characteristics which potentially make them suitable as the basis for stream-clustering; the stream can be summarised, arbitrary-shaped

clusters can be discovered, and k does not need to be specified.

2.1.4 Density-Based

Density based clustering identifies clusters as areas of high density separated by areas of low density. Density based clustering was introduced by Ester *et al.* [51] with their algorithm Density-based Spatial Clustering of Applications with Noise (DBSCAN). In DBSCAN, each point in a data-set is considered to be a core point, a density reachable point or an outlier. Each point p has an immediate neighbourhood. This neighbourhood is called its ϵ-neighbourhood, where ϵ is a user supplied unit of distance.

- p is considered a *core* point if there are at least $minPoints$ points within its ϵ-neighbourhood
- A *border* point has fewer points than $minPoints$ in its own neighbourhood but is in the ϵ-neighbourhood of a core point
- A *noise* point is any point not considered *core* or *border*
- A point q is *directly density-reachable* to a core point p if it is in p's ϵ-neighbourhood
- Two points p and q are said to be *density-reachable* if there is a path $p_1, p_i, \ldots, p_n$ with $p_1 = p$ and $p_n = q$ where each p_{i+1} is directly density reachable from p_i

Initially, a point p is selected at random and its neighbouring points are discovered. If at least $minPoints$ points are in the neighbourhood, the area is considered 'dense' and a cluster is initialised. Otherwise p is marked as noise (p might later be discovered to be in an ϵ-neighbourhood of a different point and be considered as part of that cluster). If p starts a cluster, all points in its neighbourhood are added, along with the points in their ϵ-neighbourhood. This continues recursively until all density-reachable points are found. Then, a new point is selected and the process continues until all points have been visited.

There are many variants to this original algorithm and a comprehensive overview is given in [160]. These including extensions for parallelisation [130], improved accuracy [134], and parameter estimation [54]. This idea has also been combined with a type of hierarchical clustering in OPTICS [13].

Density based methods are potentially suitable as a basis for stream clustering because they can discover arbitrary shaped clusters and k does not need to be specified a-priori. The major drawback is that only a single concept of density can be discovered. This is because the ϵ and $minPoints$ parameters are global for

each discovered cluster. Proposed solutions to this problem are mostly extensions of DBSCAN; MSDBSCAN [52], IS-DBSCAN [32], and DBSCAN-DLP [198].

2.1.5 Model-Based

Model based methods make an assumption that data has been generated by an underlying process and clustering the data involves trying to identify and model that process. The identified model defines the clusters and the likelihood of a point belonging to a cluster. Typically, a mixture of models is fitted to the data for clustering and the best model from this set is selected by some criterion.

Expected Maximisation (EM) [45] is a popular model-based clustering technique. Other model based clustering approaches are based on Neural Network variants; Self Organising Map (SOM) [103], Growing Neural Gas [135], and Self Organising Incremental Neural Networks (SOINN) [175].

The performance of model-based clustering relies on the assumed model and previous domain knowledge. In a dynamic environment these assumptions might not hold as previous concepts evolve and shift.

2.1.6 Ant-inspired

Ant inspired clustering techniques are based on the observed behaviour of ants. There is no single "ant-model" but rather a group of models based on their foraging behaviour, sorting behaviour, or their cooperative transport. The potential advantage of the ant based, swarm-intelligence approach is that clustering models and algorithms exhibit the properties of self-organisation, flexibility, and decentralisation.

Ant Colony Optimisation (ACO) [48] is perhaps the most well-known of the ant-models and is typically used as a meta-heuristic for combinatorial optimisation. In the most basic sense, this algorithm is modelled on the foraging behaviour of ants. The ants use pheromone trails to identify paths to promising foraging areas. This technique has been applied to clustering by framing the clustering problem as an optimisation problem [106, 174, 167]. As an example; each ant tries to find a cost-minimizing path, where the nodes of the path are the points in the data to be clustered. The cost of moving from data point p to q is the distance between these points. Thus, subsequent points added to the path tend to be similar to previous points on the path and the clustering process works in this way. A drawback in this approach is that there is no direct relationship between a path and a full solution (in this case, a cluster) and a further step is required. Chu *et al* [38] proposed a solution

Algorithm 1 Pick-and-Drop Model

```
1: Randomly scatter data on 2D grid
2: Randomly place ants on 2D grid
3: for t := 1 to max do
4:     ant := selected ant
5:     move ant randomly across n grid-cells
6:     g := grid-cell occupied by ant
7:     if (ant is carrying a data object) ∧ (g does not contain an item) then
8:         i := data item ant is carrying
9:         r := random number
10:        if (r ≤ P_drop) then //(Eq. 2.2)
11:            ant drops item i in grid g
12:        end if
13:    end if
14:    if (ant is not carrying) ∧ (g contains an item) then
15:        i := data item in grid g
16:        r := random number
17:        if (r ≤ P_pick) then //(Eq. 2.1)
18:            ant picks up i from grid g
19:        end if
20:    end if
21: end for
```

to this by removing connections between points that have a pheromone level below a certain threshold. Any points that remain connected by a path can be interpreted as a cluster.

A more popular approach to clustering with ants is to mimic their sorting behaviour. This behaviour has been observed in certain species of ants which sort their larvae by size and group corpses into 'cemeteries'.

This process was first modelled by Deneubourg *et al.* [46], this work introduces the 'pick-and-drop' model.

In this model, the original data points are associated with 2-dimensional (2D) points distributed randomly on a 2D grid. Ants traverse this grid and probabilistically pick-and-drop data objects and clusters emerge as a consequence of these local rules. This model was extended for data analysis [132] by introducing a dissimilarity metric to the original probabilities for picking and dropping a data object. These probabilities are based on local density in the 2D space and similarity of the data in the original d-dimensional space. The probability of an ant picking-up (P_{pick}) and dropping (P_{drop}) are defined as;

$$P_{pick}(i) = \left(\frac{k^+}{k^+ + f(i)}\right)^2 \tag{2.1}$$

$$P_{drop}(i) = \begin{cases} 2f(i) & \text{if } f(i) < k^-, \\ 1 & \text{otherwise} \end{cases} \tag{2.2}$$

where k^+ and k^- are constants, and $f(i)$ is the neighbourhood function:

$$f(i) = max\big(0, \frac{1}{\sigma^2} \sum_{j \in L} \big(1 - \frac{d(i,j)}{\alpha}\big)\big) \tag{2.3}$$

Here, $d(i,j) \in [0,1]$ is some measure of dissimilarity between two points i and j, σ^2 is the size of the local neighbourhood L (how far the ant can "see") and α is a scaling parameter in the range $[0,1]$. An overview of the process is presented in Algorithm 1.

There are a couple of major drawbacks to this approach. It is slow; each point is moved, and moved again until a predetermined number of iterations are performed. Perhaps the biggest drawback is that identified clusters can only be visually inspected and a further processing step is required to extract the identified clusters. Handl *et al.* [77] proposed a method to convert the discovered spatial embeddings into an explicit partitioning using a hierarchical clustering method. Neighbouring grid items are clustered in an agglomerative, bottom up approach.

Many modifications and extensions to the original model have been proposed. These include adaptive parameters [187], picking and dropping of more than a single item [119] and information exchange between ants [78]. In [93], the pick-and-drop model is used to initially partition the data before DBSCAN is applied.

Popular ant-based clustering methods include ACA [187] which extends the original pick-and-drop model by introducing a cooling scheme for the picking probabilities. ACAm [26] extends this by associating a short term memory with each ant. The model is further improved in ATTA [77] which uses a colony of heterogeneous ants. AntClust [114] is another popular ant-inspired clustering algorithm. However, it does not use the pick-and-drop model but is instead inspired by the chemical recognition system of ants. The idea is that ants are attracted to similar ants, data is partitioned according to this similarity between each.

Despite the aforementioned drawbacks, this model offers a potentially suitable basis for stream clustering. Discovered clusters are fluid, they can be dissolved and their contents regrouped as more data is added.

2.1.7 Summary of Traditional Methods

In this section the classical, well known clustering techniques were introduced and the potential drawbacks for their use in a streaming environment were highlighted. For example, hierarchical methods require irreversible steps during the clustering process, and partitional methods require the number of clusters (k) to be specified a-priori. Along with the limitations, some methods display characteristics that are desirable in a dynamic streaming environment. For example, density-based methods do not require k to be specified and they can discover arbitrary shaped clusters. Grid-based methods have an intrinsic summarisation method, and ant-inspired methods can create fluid clusters which can adapt as more data is added.

None of the methods described here are designed to deal with an unbounded stream of data. However, the majority of stream-clustering algorithms are based on these methods. The most common way to apply traditional batch methods to a continuous stream is to process the stream in separate chunks, or 'windows'.

2.2 Window Models

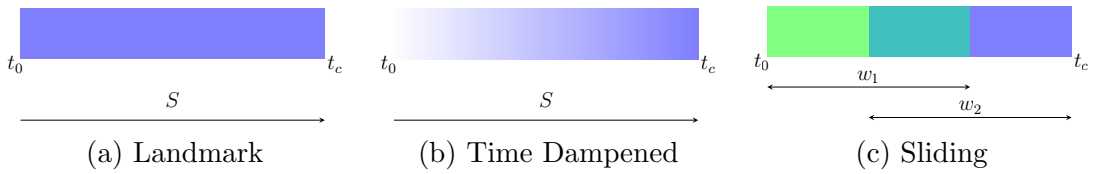
A window w is a subset of the entire stream S. A window $w[i, j] = \{x_i, x_{i+1}, \ldots, x_j\}$, where i is the start of the window and j the endpoint. There are different types of window models and the three main types are discussed here.

2.2.1 Landmark Window

With the landmark model, the entire stream from the initial starting point up until the current time t_c is considered ($w[0, t_c]$). In this model, all data is considered equally important. This is perhaps not the best approach when dealing with dynamic data as older, redundant data is treated equally as more recent, relevant data.

2.2.2 Time Dampened Window

Sometimes called the fading window. In this approach, the importance of a data sample is time-weighted. More recent data is treated as more important or relevant than older data. In this model, data be can viewed as "ageing" and when it is too old it is considered no longer relevant and ignored. Typically an exponential decay function is used to age data ($e^{-\lambda x_{age}}$).



(a) Landmark (b) Time Dampened (c) Sliding

Figure 2.1: Different windowing models. Stream S begins at t_0 and runs until current time t_c. All data considered under landmark model (a), greater importance given to recent data in the time-dampened model (b). Overlapping windows (w_i) in sliding window model (c)

2.2.3 Sliding Window

In the sliding window model we are interested in only the most recent subset of the stream. This subset w could be a fixed amount of incoming points or it could be a fixed period of time. The size of w is an important consideration. If it is too large, redundant data will persist too long and affect the quality of analysis, conversely if it is too small, data will age too quickly before reliable models can be built or clusters discovered. Techniques for an *adaptive* window size have been proposed [24, 115], here when accuracy is high the window size can be extended. If the accuracy starts to decrease, this suggests a change and the window size is reduced. These techniques have been applied to supervised classification environments but could be extended to unsupervised environment with an appropriate change detection method. The differences between each of these three models is illustrated in Figure 2.1

2.3 Clustering Data Streams

This section presents an overview of the most influential ideas in stream-clustering along with recent advancements in the field. Stream clustering algorithms typically extend the traditional methods but often adopt a combination of approaches, for example density-based and grid-based hybrids.

2.3.1 Partitional & Hierarchical

STREAM [147] was one of the first proposals for stream-clustering. A stream is processed in fixed-sized windows and each window is partitioned using a k-mediods variant. The authors proposed *LSearch*, which incorporates a local-search function to speed up the convergence of the original K-mediods method. Once clusters have been discovered, their mediods are weighted based on how many points are present in the cluster. Only these weighted points are retained and carried over to the

next window, the remaining points are deleted. This approach suffers from the traditional problems of partitional methods but allows for a stream-processing due to the memory efficiency of the summarisation method.

BIRCH [204] is another early stream-clustering algorithm that uses a different summarisation method; the Cluster Feature Vector (CF). A CF for a cluster containing N points $\{\vec{X}_i\}$, $i = \{1, ..., N\}$ is described using three components: the number of data points the micro-cluster contains (N), the Linear Sum (LS) of each feature (i.e.,$\sum_{i=1}^{N} \vec{X}_i$), the Squared Sum (SS) of each feature (i.e., $\sum_{i=1}^{N} \vec{X}_i^2$). So, the CF for cluster $c_i = [N, LS, SS]$. From this triplet, the centre c and radius r of the cluster can be found:

$$c = \frac{LS}{N} \tag{2.4}$$

$$r = \sqrt{\frac{SS}{N} - \left(\frac{LS}{N}\right)^2} \tag{2.5}$$

CF vectors are incremental (points can be added and the CF updated) and additive (two clusters can merge into one). These characteristics are hugely desirable for processing a stream. BIRCH works in a single pass of the data and CF vectors are stored in a tree structure called a CF-Tree. A predetermined number (k) of clusters are extracted from this tree using an agglomerative approach.

CluStream [1] introduces the micro-cluster as a summarisation method. A micro-cluster is a temporal extension to the CF. Each point p_i in the cluster has an associated time stamp T_i. The sum of these time stamps is maintained in the micro-cluster summary. A micro-cluster $mc_i = [N, LS, SS, T]$. CluStream also introduces a widely adopted two-phase approach to stream clustering; the online/offline model whereby data is summarised online and the summarised data is clustered offline. CluStream uses this approach combined with the temporal aspect of micro-clusters to discover clusters at different levels of granularity. This granularity is specified by a user parameter h, the horizon. Micro-clusters within the window $[t_c–h, t_c]$ are clustered offline using the k-means partitional method. CluStream has been extended in HP-Stream [2] to handle high-dimensional data, and in SWClustering [207] to better approximate concept drift.

ClusTree [112] uses micro-clusters as a summarisation method but alters CluStream's approach by recording only the latest time-stamp of a cluster (as opposed to the sum of all time-stamps). Micro-clusters decay at an exponential rate and if no new data has been inserted into a micro-cluster for some time it is considered no longer useful and can be deleted. However, if data is added, the time-stamp is updated and the micro-cluster is considered recent and relevant. ClusTree also adopts

the two-phase approach and stores micro-clusters in an online tree-structure. It is more sensitive to time-constraints in a stream and allows for an 'any-time' insert of new data into the tree. An incoming point can be inserted into the most appropriate micro-cluster if there is sufficient time to traverse the tree, if there is not a lot of time (high velocity streams) an approximation is made. ClusTree returns clusters using the k-means algorithm in the off-line stage and was shown to perform better than CluStream.

The methods introduced here use micro-clusters as the summarisation method, other approaches use micro-clusters as both the summarisation method *and* the clustering mechanism.

2.3.2 Density-Based

DenStream [31] is one of the most influential density based methods. It extends the micro-cluster (mc) in two ways; the linear sum and squared sum of the contents are time-weighted using an exponential fading function. The data becomes less relevant the older it gets. The second extension is to introduce an upper limit on the size of its radius. This limit is denoted as ϵ and performs the same function as the ϵ-neighbourhood in DBSCAN (Section 2.1.4). An mc is considered dense if its weight is above a user-defined threshold μ and its radius is below ϵ. Such an mc is referred to as a *core* micro-cluster.

Micro-clusters can also be *potential* or *outlier*. A potential mc is one whose weight is above $\beta\mu$ where β is a parameter used to determine the threshold relative to μ. Outlier mcs have a weight less than $\beta\mu$. Because the data is time-weighted, new data inserted into an mc will increase its weight and *outliers* can be promoted to *potential* and eventually *core*. Similarly, if no new data is added, core-mcs can get demoted to potential, outlier, and eventually removed if their weight falls below a certain threshold. When an online point p arrives, DenStream tries to merge p with an existing potential mc pmc_i, if no suitable one is found (or if, by inserting p into pmc_i the radius of pmc_i exceeds ϵ) p tries to merge with an existing outlier mc omc_i, if this is unsuccessful, a new outlier omc_j is created with p.

Periodically, the weights of each mc is updated, mcs can change from core to potential (or vice versa) and redundant mcs can be removed. When a clustering request is made core-mcs are identified and a clustering is performed offline using DBSCAN.

There are many variants and extensions to DenStream. C-Denstream [166] allows for background knowledge to be incorporated in the clustering process using instance level constraints (Must-Link, etc.). R-Denstream [124] ('R' for retrospective)

Algorithm 2 CEDAS

```
read point p
if (can add p to existing micro-cluster mc_i) then
    add p to mc_i
    mc_i energy = 1
    if (p added to mc_i shell region) then
        update mc_i centre
    end if
else
    create new micro-cluster mc_j from p
end if
for (all micro-clusters mc_i) do
    decay mc_i energy
    if (mc_i energy ≤ 0) then
        delete mc_i
    end if
end for
if (any micro-cluster updated) then
    check for new intersections
    if (new intersections) then
        Update macro-assignments for all linked clusters
    end if
end if
```

introduces a third phase in addition to the online/offline phases. Here, outlier mcs which would ordinarily be deleted are retained and revisited later. HDenstream [126] accommodates categorical data and SWClustering [207] which stores the outlier and potential micro-clusters in a space-efficient histogram structure. All of these extensions to DenStream use the two-phase approach. However, it has been observed [66, 189] that the offline clustering phase is computationally expensive and is only performed when requested, this presents a trade-off between frequent requests to better discover changes in the stream and infrequent requests in order to reduce computational overheads.

To overcome this, the two phases of DenStream were merged into a single online phase in *FlockStream* [66]. This algorithm adopts the concepts of time-weighted *core* micro-clusters and *non-core* micro-clusters introduced in DenStream but the clustering process is modelled on the observed flocking behaviour of birds proposed in Reynolds' Boids algorithm [162]. This process is similar to the ant-clustering methods described in Section 2.1.6, in that data is mapped to a 2D toroidal grid. Flocking is an emergent behaviour of each agent in the grid following local rules. A dissimilarity metric (Euclidean distance between points in the original data space)

creates different flocks as similar micro-clusters cluster and avoid groups of dissimilar micro-clusters.

Another recently proposed algorithm for clustering in one single online phase is CEDAS [87]. CEDAS creates a graph structure to represent the relationship between micro-clusters. A micro-cluster is divided into two regions; the inner 'kernal' region (area between centre of mc and $r/2$) and the outer 'shell' (area between r and $r/2$), where r is the mc's radius. Micro-clusters mc_p and mc_q are connected if there is an intersection between mc_p's kernal region and mc_g's shell. This intersection is described as an edge linking them. All mcs that are connected form the macro-cluster. Each mc has an 'energy' level which fades linearly if no new data is added. If it reaches zero, it is deleted. Whenever an mc is updated (created, deleted, or a point added to a shell-region), CEDAS searches for new intersection and the graph structure is updated. The CEDAS process is sketched in Algorithm 2

Density-and-Grid hybrids are proposed in D-Stream [36] and MR-Stream [189]. MR-Stream divides the data space into a tree of grids, the size of the grids decrease (increased resolution) the further down the tree. The cluster grids are updated online and clustered offline. The offline also performs a tree-pruning stage. In D-Stream each dimension is divided into n segments, this creates a grid of hyper-rectangular cells. Data is assigned to an appropriate grid-cell and each cell is summarised. A fading window is used to age data and sparse grid-cells are periodically removed. With a clustering request, clusters are discovered offline. Neighbouring dense grid-cells form a cluster.

The density based methods described so far all share a common shortcoming: an inability to detect clusters of varying densities. The reason for this restriction to a single concept of density is the use of global parameters for each cluster. The ϵ-neighbourhood defines the maximum radius for a micro-cluster and the *minPoints* parameter defines how many points a micro-cluster must contain before it can be defined as 'dense'. These parameters apply to every discovered cluster. A global ϵ imposes limitations on the *type* of clusters that can be discovered. For example, embedded or overlapping clusters will be missed if only using a single concept of density. Recently, there have been a couple of proposals to address this problem; AD-Stream [47] and MuDi [12].

AD-Stream (Adaptive Density Stream) uses the two-phase approach. Online points are mapped to a grid and grid contents are summarised. Using a combination of a weighted similarity matrix and associated eigenvectors, grid areas of varying densities can be discovered in an offline step. MuDi also uses a grid structure and a two-phase approach. A grid structure is created at a user defined level of gran-

Algorithm 3 Competitive Neural Network Model

```
Read point x_i
Find closest node p to x_i
Find second closest node q to x_i
Assign x_i to p
if (p and q are connected by edge e) then
    e_age = 0
else
    create edge e between p and q
    e_age = 0
end if
if (Time to add new node) then
    Find node z with largest cumulative error
    Find neighbour ẑ of z with largest cumulative error
    Add a new node r half way between z and ẑ
    Delete original edge between z and ẑ
    Insert edge connecting z and r
    Insert edge connecting r and ẑ
end if
for (each edge e_i) do
    increase age of e_i
    if (e_{i_age} > max) then
        delete e_i
    end if
end for
Delete any isolated node
```

ularity and, separately, a set of micro-clusters are maintained. First, an incoming point p tries to merge with an existing mc. If this is not possible, p is mapped to an appropriate grid cell. The contents of each grid cell are time-weighted, if a grid weight is above a certain threshold its contents are merged into a new micro-cluster. If the weight falls below a certain threshold, the grid contents are deleted. When a clustering request is made, the micro-clusters are clustered offline using an extension of DBSCAN called M-DBSCAN. This variant replaces the ϵ neighbourhood of DBSCAN with a concept of local cluster density, called core-neighbouring. Micro-clusters are added to existing clusters if they have similar mean values with some acceptable differences in standard deviation. This allows for multi-density clusters, however this algorithm requires a lot of very sensitive parameters; the grid granularity is crucial and the $minPonts$ parameter in M-DBSCAN is sensitive. Both of these parameters can greatly affect clustering performance. Very sensitive parameters like this are not ideal in a stream. It might be possible to tune them on a subset of the stream but in a dynamic environment a good set of parameters at time t might not

be the best at $t + \delta$.

2.3.3 Model-Based

Competitive Learning is a common approach to model-based stream clustering. Most are extensions to Neural-Network inspired clustering methods like the Self Organising Map (SOM), Growing Neural Gas (GNG) and the Self Organising Incremental Neural Network (SOINN). These methods broadly adopt the same approach; during the online step an incoming point x_i triggers a competition between existing nodes. The "winner" q is the closest node to x_i, p is the "second winner". x_i is assigned to q and an edge is created between p and q (or if an edge already exists, it is strengthened). Nodes and edges are time-weighted and are removed if their weight gets too low. Isolated nodes can be deleted and periodically, a new node is inserted between two existing nodes that have the largest accumulated error (farthest from previously seen points in the stream). This process is outlined in Algorithm 3. SWEM [42] is an EM-Based method which uses sliding window model. Each model has a mean, weight and a covariance matrix. In the first window, EM is used to obtain the parameters for each model. These parameters are refined in subsequent windows. If parameters discovered in latest window are significantly different than previous windo,w SWEM abandons models with the largest variance and creates new models while merging components that are similar.

A common problem with this approach is the difficultly in extracting the clusters from the created graph structure [65, 200]. AING [27] and G-Stream [71] overcome this problem, both by defining each node as a cluster. DenSOINN [200] treats each node as a micro-cluster, when a cluster request is made, these micro-clusters are clustered offline using a method based on density connectivity similar to DBSCAN.

2.3.4 Ant-Based

There are almost no ant-based stream clustering algorithms. Cl-AntInc [136] is proposed as a partitional method for clustering streams using sliding windows of different sizes. Initially, a window is clustered using k-means and the initial k-means partitions are refined using a model of ants following pheromone trails between each of the k clusters. When a new window arrives, the entire contents of the window is treated as a new cluster and one by one the contents of the new cluster are assigned to the existing k clusters based on similarity of each point ('ant') with the cluster. This repeats until all ants have been assigned and the latest cluster is deleted, maintaining the original k number of clusters. Not much consideration is given

Table 2.1: Overview of Stream-Clustering Algorithms

Algorithm	Clustering Method	Windowing Method	k required	Processing
[204] Birch (1996)	Hierarchical	Landmark	Yes	two-phase
[147] Stream (2002)	Partitional	Sliding	Yes	two-phase
[1] CluStream (2003)	Partitional	Sliding	Yes	two-phase
[112] CluStree (2011)	partitional	Time Dampened	Yes	two-phase
[136] CL-AntInc (2015)	Partitional & Ant	Sliding	Yes	two-phase
[31] DenStream (2006)	Density	Time-Dampened	No	two-phase
[66] FlockStream (2011)	Density	Time-Dampened	No	online
[47] AD-Stream (2016)	Density	Time-Dampened	No	two-phase
[87] CEDAS (2017)	Density& Graph	Time-Dampened	No	online
[189] MR-Stream (2009)	Density& Grid	Time-Dampened	No	two-phase
[12] MuDi (2016)	Density & Grid	Time-Dampened	No	two-phase
[42] SWEM (2009)	Model	Time-Dampened	No	online
[71] G-Stream (2016)	Model	Time-Dampened	No	online
[200] denSOINN (2018)	Model	Time-Dampened	No	two-phase

to memory constraints and this method suffers from the drawbacks of traditional partitional methods.

2.3.5 Summary of Stream-Clustering Methods

The main stream-clustering algorithms discussed in this section are presented in Table 2.1. These methods are largely extensions of traditional methods coupled with windowing methods to deal with the memory constraints of an unbounded stream. Despite there being a large body of research on ant-based clustering with batch data, this has not transferred to stream-clustering. Density based clustering methods appear to be the most common in the literature. They offer advantages in that k does not need to be specified and arbitrary shaped clusters can be discovered, allowing noise and outliers to be better managed. Most density approaches use the micro-cluster as both the clustering mechanism and as the summarisation method. A micro-cluster allows a number of similar points to be represented by one point and furthermore, has the attractive characters of being increment-able and additive. This allows micro-clusters to adapt and change with new data. Density methods are common in hybrid approaches; they are used in combination with graph, grid and model-based methods.

2.4 Classification in Non-stationary Streams

Classification is the task of identifying to which set of *pre-defined categories* a new observation belongs to. This pre-defined set of known categories is what differentiates the classification task from the clustering task.

There are many classification techniques designed for the traditional batch setting. Artificial Neural Networks [194], Baysian Classifiers [18], Nearest Neighbour [165], Support Vector Machines [186], Decision Trees [168] and others [107]. Traditionally, a finite set of labelled data is collected, this data is then split into a training set and a testing set. A model is created on the training set and verified on the testing set. These models assume that subsequent unseen data is sampled from the same fixed, distribution as the original data-set. The assumption is that if the model performed well in the testing phase it should continue with roughly the same reliability on all future samples. Of course, this is not true if the underlying distribution is non-stationary. Traditional classification methods form the basis of stream classification, often in conjunction (as in stream-clustering) with windowing methods.

Ensemble methods combine multiple classifiers into a single (hopefully stronger) classifier. Any combination of traditional classifiers can be combined and each member's prediction contributes to the ensemble's final decision. The intuition behind this is that there is no specific algorithm that will work best in all cases; each model will have its own strengths and this can be exploited by a diverse group of base-classifiers. Popular methods for creating an ensemble are bagging and boosting [43]. Bagging creates multiple different training-sets from the original training-set by performing sampling with replacement. Each base classifier is trained on a unique training-set and the ensemble's output is determined by majority voting from all base-classifiers. Boosting incrementally builds an ensemble by training each new classifier on instances that the previous model misclassified and the ensemble's prediction is again based on majority voting. Adaboost (adaptive boosting) weighs each sample so that training samples correctly classified by the previous base-learner are weighted less than samples which are misclassified. Base-classifiers are also weighted based on their accuracy and the ensemble outputs a prediction based on weighted voting of its member's predictions.

Ensembles provide a natural mechanism for adapting to changes in the data (add a new member) and forgetting irrelevant data (remove a member), in doing so ensembles provide a good solution to the *stability-plasticity dilemma* [73], the ability of a model to retain existing knowledge *or* learn new knowledge but not to be able to both equally well. Ensembles also have the advantage that in the presence of change,

the entire model doesn't need to be retrained, just the subset of the ensemble that is affected.

This section presents an overview of classification in non-stationary streams. The section is divided into fully supervised methods, and methods which assume a scarcity of labels.

2.4.1 Supervised Classification

Supervised methods can be broken into two approaches, single model and ensemble. These can be broken down further into *active* and *passive* approaches. Both active and passive approaches maintain an up-to-date model which can respond to change however, each uses a different method. Active methods use a two-phase "detect and react" [6] approach, whereby a mechanism (separate from the classifier) monitors the stream for change and if change is signalled the current model is updated with the latest samples from the stream. Passive approaches do not monitor the stream for change, instead they accept that change will happen and continuously adapt, updating the model with newly arriving data. Single models and ensembles are presented from both active and passive perspectives.

Single Model

Active approaches adopt the "detect and react" approach.

Detect: Change detection methods typically operate on characteristics or features extracted from the stream rather than the original raw features, for example statistical measures like the mean and variance of a sequence or the classifier's performance on the stream. Hypotheses Testing (HT) methods create a hypotheses of the data and test this hypothesis according to some predetermined confidence threshold. The authors in [151] use a distance metric (in this case the normalised Kolmogorov-Smirnov distance) to measure the differences between density functions estimated on the training data and a window of recent data. Change is signalled if this distance measure exceeds the threshold. Change Point Methods (CPM) also operate on fixed size windows, they attempt to find a change-point in the window by examining all possible partitions of the data. This is very computationally demanding, however CPM can not only detect change but also identify the time-step that change has occurred [81]. Sequential Hypothesis Tests (SHT) can operate on a sample-by-sample basis (as opposed to a fixed subset of the stream). A sequence is analysed until the mechanism is confident enough to make the decision that change has occurred. Example SHTs include the repeated significance test [14] and the

sequential probability ratio test [188].

These methods are typically computationally demanding but offer theoretical guarantees that change (if it occurs) will be detected. Change Detection Tests (CDT) differ in that they are typically more resource- efficient but can not guarantee a control of false positive rates. CDT commonly make decisions based on the observed classification error of the model. A threshold is defined and if performance drops below this threshold then change is signalled. Often, non-overlapping sliding windows are considered [79]. Change is triggered if the validation error on the latest window is significantly different from a random window that has been processed in the past. Or the error on the latest window is compared with the error on the initial training data [39]. Another CDT is proposed based on the Cumulative Sum (CUSUM) of the error [5]. Here the sample mean of the error is monitored over time and statistical variations signal change. CDTs are straightforward to implement but the challenge is in setting an appropriate threshold. If this value is too low there will be many false positives but if too high actual change will be unnoticed.

ADWIN [22] is a popular change detector that tracks the average of a stream. ADWIN keeps a variable-length window of processed samples, with the hypothesis that "there has been no change in the average value inside the window" [22]. The largest window that can be maintained which is consistent with this hypothesis is maintained. Older sections of the window are ignored only if there is sufficient evidence that the average value of that window differs from the rest of the window. The raw values from the window are not maintained, instead the features are compressed using a variant of the exponential histogram technique, these compressed features constitute the average.

Adapt: Once a change is signalled, the classifier is updated. Typically, the old classifier is abandoned and a new classifier is trained on recent data. The Very Fast Decision Tree (VFDT) [49] is a commonly used single model due to its ability to quickly process large amounts of data [5, 39, 67, 22]. The VFDT is an incremental algorithm that exploits the fact that, when creating the tree structure, the optimal splitting value can be identified using only a small sub-sample of the data. This idea is theoretically proven using the Hoeffding bound (VFDTs are often called 'Hoeffding Trees').

Once change is detected a new model is trained on recent data. Sliding windows are used in [67, 22, 7, 21], here only the most recent window is used to train the new classifier and points outside this window are discarded. As previously discussed in Section 2.2, it is a challenge to find an appropriate window size; the dilemma is selecting a length that provides enough training samples for the new classifier while

ensuring that only relevant samples are used. This trade-off is mitigated in models which use a time-dampened window [108, 40, 102]. Here, all available samples are considered but with greater relevance given to recent data. Weights decrease linearly in [108], polynomially and exponentially in [40].

The main drawback with these methods is that sample points cannot be discarded or summarised once they have been processed because they will likely be required as training points later in the stream. If the window is large, or if very distant points are considered, there is potential for memory problems.

Passive approaches assume change in the stream and continuously update the model. The VFDT was extended to passively handle change with the CVFDT (Concept Adapting Very Fast Decision Tree) [86]. Here, a standard VFDT is maintained and a sliding window is used to process the stream. As in the original algorithm, the tree is updated incrementally at each window. However the CVFDT examines each node looking for places where a split has been made in the past but which would not be selected in the current window. This could be a signal of change. If such a split is detected an alternate subtree is grown at this decision point. If this new subtree becomes more accurate than the original, then change is confirmed and the new tree replaces the original; otherwise, the original remains. Hoeffding Option Trees [70] extend a VFDT by incorporating additional option nodes that allow several Hoeffding tests to be applied at each split. This creates a single tree structure that represents multiple different trees. This is further extended with *adaptive* Hoeffding trees which, at each node, also store an estimation of the current error [149].

Single Model classifiers generally require lower computation costs than ensemble methods, which makes them a good solution in high-velocity data streams. Despite this, ensemble approaches appear to be more common in the literature.

Ensemble

Ensemble methods also can be active or passive in how they deal with change. **Passive** methods are the most popular and are discussed first.

The Streaming Ensemble Algorithm (SEA) [180] is one the earliest work on supervised ensembles and introduced a general framework (outlined in Algorithm 4) which is common in the literature [191, 53, 140]. A stream is processed in fixed-sized sliding windows and at each window a new classifier is trained and added to the ensemble. Once the ensemble reaches a predefined size, the weakest (according to some quality measure) base-learner is removed. SEA favours classifiers that correctly classify samples which are almost undecided by the ensemble. This maintains diversity and helps prevent over-fitting. The ensemble outputs a prediction based

Algorithm 4 Traditional Window-Based Ensemble Method

1: Read current window w_c in stream S
2: Train new classifier ϕ_{new} on w_c
3: **for** (each classifier ϕ_i in ensemble E) **do**
4: evaluate ϕ_i on w_c
5: **end for**
6: Find worst performer ϕ_{worst} in E
7: **if** ($|E| < maxSize$) **then**
8: add ϕ_{new} to E
9: **else**
10: replace ϕ_{worst} with ϕ_{new}
11: **end if**

on majority voting. Accuracy Weighted Ensemble (AWE) [191] follows a similar structure but assigns weights to each classifier based on their prediction error on the latest window. The idea being that the latest samples are most representative of the current concepts in the stream. Classifiers that perform worse than a random classifier are removed from the ensemble. The drawbacks to this approach are that usually, k-fold cross validation is used to create each classifier which can be costly and also, as is frequently the problem with fixed-sized windows, determining window length is a challenge. In this instance, the problem is the trade-off between a small enough window that can detect and react to change and a large enough window that contains enough samples to adequately train a classifier.

Scholz and Klinkenberg [172] propose a method to overcome this trade-off by using incremental learners as the base-classifiers. The latest window w_c is read and the most recent classifier (the classifier trained at (w_{c-1})) is updated with the samples in w_c. In parallel, a brand new classifier is trained on w_c. The ensemble adds the best performing of these two (highest prediction accuracy on the latest window). The advantage here is that, in times of stability, the most recent classifier can learn samples from multiple windows. Only when a change occurs, will a new classifier be added. The Accuracy Updated Ensemble (AUE) [28] also uses incremental learners, VFDTs. At each window, each component classifier is incrementally updated with a subset of the window, majority voting is used to determine the output and periodically, the weakest learners are pruned to maintain the ensemble size. Yet another approach is presented in the Weighted Ageing Ensemble (WAE) [197]. Here, classifiers are maintained based on their diversity aswell as their prediction accuracy. The ensemble prediction is a weighted vote from each member with greater weight given to classifiers that have been in the ensemble the longest coupled with their recent prediction accuracy.

The ensemble methods discussed so far have all processed a stream in fixed-sized windows, or chunks. Other methods are online and read the stream on a sample-per-sample basis.

An influential online ensemble is the Dynamic Weighted Majority (DWM) ensemble [105]. This extends the Weighted Majority Algorithm [129] proposed for stationary streams. In DWM, each member has an associated weight and this weight decreases with every incorrect prediction. The weight decreases by a multiplicative constant β ($0 \leq \beta \leq 1$).

These weights change over time and base learners with a higher weight are considered to be more 'expert' on the current portion of the stream. After every n samples are processed, the classifier weights are updated and a classifier can be added or removed. Points which have been misclassified by the ensemble are retained and used to train a new classifier from scratch. Existing classifiers with a low weight are assumed to be no longer helpful and are removed. The Concept Drift Committee (CDC) [179] follows a similar model but member's weights are proportional to their accuracy over the last n training samples.

The drawback of the DWM ensemble is that the choice of n can be sensitive. Addictive Expert Ensembles (AddExp) [99] are similar to DWM however the ensemble is updated each time a point is misclassified. This removes the need to specify n. The oldest classifiers are pruned whenever the ensemble reaches a predefined size. Similar approaches are presented in Winnow or Hedge β [113]

Active Ensemble Methods are not as common as passive ones. Diversity for Dealing With Drifts (DDD) was proposed in [141]. The intuition behind this is that ensembles with a low variance are good in times of stability, however in times of change an ensemble with high variance is preferable. DDD maintains two ensembles concurrently: one with low variance which is used for learning and predicting, and a separate high-variance one which is used to learn but not make predictions. If a drift is detected, then both ensembles are combined to make predictions. Weighted voting is used and each ensemble is weighted by its prequential accuracy since the time step in which drift was detected. This approach is shown to be robust to change and false-alarms. However it has the obvious drawback of requiring two ensembles to be maintained.

The original Random Forest algorithm is an ensemble of decision trees created by bootstrapping a training set and is designed for stationary data. An extension is proposed in Adaptive Random Forest (ARF) [72] to handle non-stationary streams. ARF is online and uses an ensemble of VFDTs combined with a novel approach to drift detection. Any potential change is first signalled as a 'warning'. During

Algorithm 5 Adaptive Random Forest

```
read stream latest point p from stream S
(x, y) := p
for each classifier φ_i in ensemble E do
    Prediction ŷ_i := φ(x_i)
    Update φ_i with y
    if (drift warning) then
        φ_{i_b} := new classifier
        add φ_{i_b} to background classifiers B
    end if
    if (drift detected) then
        φ_i = φ_{i_b}
    end if
end for
for each φ_{i_b} in B do
    update φ_{i_b} with y
end for
Return majority ŷ_i
```

a warning, a brand new 'background' tree is created and trained on all incoming instances, this background tree only learns and does not affect the ensemble's decision making. If the initial warning turns out to be actual change then the tree that signalled the warning is replaced by its respective background tree. The rough pseudo-code for ARF is outlined in Algorithm 5. ARF is not bound to any specific change detector though the authors used ADWIN.

Online versions of bagging and boosting have been proposed in [149], these methods have been used alongside ADWIN for ensemble maintenance: OzaBoost and BagAdwin in [23]. OzaBoost uses an ensemble of VFDTs which is maintained using online boosting and BagAdwin also employs VFDTs as the base classifiers in an ensemble maintained using online bagging whenever ADWIN detects a change. Iterative Boosting Ensemble was proposed in [94], where a window is read and incremental learners are trained using boosting.

Ensembles of micro-*classifiers* have been proposed [3, 8]. A micro-classifier is a supervised version of the micro-cluster (see Section 2.3.2). In addition to the summary information recorded in a micro-cluster, a class label is associated. The approach was originally outlined in [3], which is a supervised version of the CluStream algorithm for clustering data streams [1]. Initially, a predefined number training points are collected and labelled. These training points are separated into their respective classes and k-means clustering is performed on each class. Each micro-cluster $mc_i \in C$ ($|C| = j * k$, where j is the number of classes in the training set) is

Table 2.2: Overview of Supervised Ensemble Methods

Algorithm	Detection Method	Processing Method
[23] OzaBoost (2009)	Active	Online
[23] BagAdwin (2009)	Active	Online
[141] DDD (2012)	Active	Online
[72] ARF (2017)	Active	Online
[94] IBS (2019)	Active	Chunk
[105] DWM (2007)	Passive	Online
[179] CDC (2003)	Passive	Online
[3] On Demand Stream (2004)	Passive	Online
[8] CLAM (2013)	Passive	Chunk
[180] SEA (2001)	Passive	Chunk
[191] AWE (2003)	Passive	Chunk
[172] KBS (2005)	Passive	Chunk
[28] AUE (2014)	Passive	Chunk
[197] WAE (2013)	Passive	Chunk

treated as a micro-classifier $m\phi_i$. Incoming points are classified and then clustered. First a nearest neighbour method is applied. The closest micro-classifier $m\phi_i$ to incoming point x_i is identified and the predicted label of x_i is the label associated with $m\phi_i$. The point x_i is then either inserted into $m\phi_i$ (if the distance between the two is within a maximum boundary) or a new micro-classifier $m\phi_j$ is created with the true label y_i of x_i . This method maintains a fixed number of micro-classifiers. If a new micro-classifier is created, an existing one must be deleted (one with a time-stamp older than a given threshold) or if two micro-classifiers are close enough in the feature space (and both have the same label) they can be merged. Similar approaches are used in [8, 9]. A summary of the supervised ensemble techniques for classifying non-stationary streams is presented in Table 2.2.

2.4.2 Classification with a Scarcity of Labels

Methods for learning in non-stationary environments with a scarcity of labels (SoL) typically fall into two general approaches; active learning (AL) and semi-supervised learning (SSL). Both of these approaches are well established in the field of machine learning but have only recently been applied to non-stationary environments [44].

An AL mechanism identifies the most important instances in an unlabelled set and requests labels for those instances from an "oracle", usually a human expert. This requires manual intervention but greatly reduces labelling costs. Usually these approaches work within a label budget so the more labels that can be provided to

the algorithm the better the results. AL assumes that if a request is made a label will be made available immediately.

SSL methods assume that a set of n samples $x_1, \ldots, x_n \in X$ with their corresponding class labels $y_1, \ldots, y_n \in Y$ are available in addition to u unlabelled samples $x_{n+1}, \ldots, x_{n+u} \in X$. SSL attempts to build classifiers using the combined data set, as opposed to supervised learning which would use the first n labelled samples or unsupervised learning which would use the u unlabelled samples. Non-stationary classification with a scarcity of labels is discussed from these two perspectives.

Active Learning in non-stationary streams

Traditional AL methods adopt a 'pool-based' approach whereby the interesting samples are selected by examining all historical data. Obviously, this is not feasible in a stream and often sliding windows are used [208, 127, 61, 128, 88]. In these cases, each window is treated as a static batch and samples are selected in each window. A passive ensemble approach with sliding windows was proposed in [208]. A new classifier ϕ_i is trained on each new window w_i and is added to the existing ensemble E. A minimum variance approach is used to select which samples in w_i should be labelled; a point $x \in w_c$ is classified by every base classifier in E. If the variance is low (if the base-classifiers agree on their predictions) the corresponding label for x is not requested. If the variance is high, then a label is requested. In [127] a passive windowing approach was implemented to extend a static AL method proposed in [199]. Here, an SVM is used as a classifier and during each window samples which are closest to the hyperplane are selected for labelling. Confidence Distribution Batch Detection (CDBC) [128] measures the difference between the distribution of the classifier outputs in the most recent window and a reference window. If this difference exceeds a predetermined threshold then a change is triggered and all instances from this point on are collected into a training window of size n. When these points are collected, labels are requested for the entire window and a new classifier is trained. The authors use a SVM with a linear kernel. This approach was shown to be accurate and respond well to change. However, the value of n is important and in comparison with other AL approaches it can require a relativity large labelling budget, especially if the stream is volatile and a lot of change occurs. A decision tree is used as a classifier in [61], drift is detected if there is sufficient change in the summary statistics maintained at each node in the tree. If change is suspected, n random samples from the latest window are selected and the classifier is evaluated on these n labelled-samples. If the classifier's performance is below a threshold θ the classifier is discarded and a new classifier is trained on the latest samples.

Algorithm 6 The Active Classifier

```
1: Read latest unlabelled point x
2: if (requested labels < max labels ) then
3:     if (AL strategy requests label) then
4:         y = true label of x
5:         request labels++
6:         Update learner φ with (x, y)
7:         if (Background learner φ_b exists) then
8:             Update φ_b with (x, y)
9:         else if (warning detected) then
10:            Start new background learner φ_b
11:        end if
12:        if (Change detected) then
13:            Replace φ with φ_b
14:        end if
15:    end if
16: end if
```

The approaches mentioned so far are extensions to AL approaches in stationary environment and do not fully consider the potential impact of concept drift in a data-stream. Specifically, these approaches respond to virtual drift, a change in $P(x)$. Changes which happen away from the decision boundary will be missed, for example, when concept evolution occurs or during real drift when there is a change in $P(y|x)$ but no change in P(x). In these instances we can say that the full instance space has not been inspected for change. This problem is addressed in the Active Classifier [209]. In this study, a generic framework for handling a stream (online and incrementally, with an active drift detector) is proposed along with a number of different AL strategies to work within this framework. The general framework is outlined in Algorithm 6 and is similar to the approach used by the Adaptive Random Forest described in the previous subsection. Change is first signalled as a warning, when a warning is initiated a background learner is created and learns in parallel with the main classifier, though it does not make any predictions. If the warning is true and change is detected, the original classifier is replaced with the new one.

In the Active Classifier, a very tight constraint on the labelling budget is imposed; only a finite amount of labels are available to use over the entire, potentially infinite stream, unlike in window based approaches where each window had n label-requests. This is perhaps an unrealistic constraint. However, the authors proposed an 'approximate spending estimate" to keep track of how many labels have been requested relative to the amount of stream that has been processed. Five AL strate-

gies are proposed and each can be used in the general framework (line 3, Algorithm 6). 1) Random: A label is requested with probability p. 2) Fixed Uncertainty: this strategy is based on the posterior probability estimates of the classifier ϕ. If the certainty is below a threshold θ, a label is requested. 3) Variable Uncertainty: the sample with the least certainty within a given time interval is selected. 4) Uncertainty with Randomisation: request labels that are close to the decision boundary. 5) Split: a combination of the random strategy with variable uncertainty. The Active Classifier obtains good results with a very small labelling budget. The split strategy was shown to be the best AL approach because it effectively spread the labelling efforts across the entire instance space.

Another approach that covers the whole space is proposed in Active Cluster Learning (ACL-Stream) [88]. ACL-Stream uses a single incremental classifier and reads the stream in windows. At each window samples are selected for labelling based on clusters discovered (using k-means) in the window. Each discovered cluster $c_i \in C$ contains n clustered points $x_i, \ldots, x_n$. Samples are selected for labelling in a two-step process. First, the macro step: every point is classified by the classifier ϕ which has been trained on all labelled data so far. Clusters are ranked according to their homogeneity. If the points in a cluster all have the same predicted label then the cluster is homogeneous and considered less interesting. However, if a cluster contains many different classes (as predicted by ϕ) then it is considered an area of interest and is ranked highly. In the next step, the micro step, the contents of each cluster is ranked by a function of its distance from the centre of its respective cluster and the classifier's uncertainty of its label. Once all instances and clusters have been ranked, the top b samples are selected (where b is the labelling budget). These labelled samples are used to incrementally train ϕ.

Clustering and classification in non-stationary streams are more typically applied in semi-supervised methods.

Semi-Supervised Learning in non-stationary streams

The micro-cluster introduced in stream-clustering [1] as a fully unsupervised mechanism was used successfully as a supervised classifier in On Demand Stream [3] so it is logical to expect the micro-classifier/ micro-cluster structure to be used in the semi-supervised setting (in this section micro-classifier and micro-cluster can be used interchangeably). An early example of its application is the Semi-Supervised Ensemble Approach (SSEA) [125]. The stream is processed in fixed-sized windows, and each window w_i is split into j partitions, where j is the number of known classes. k-means clustering is used to create k clusters in each of the j partitions. Each clus-

ter is summarised by its centre, radius, number of instances, and class label. This summary information is the micro-classifier $m\phi$. Formally, a model M is created at each window where $M = \{m_1, \ldots, m_j\}$ and m_i represents a set of micro-classifiers specific to each known class $m_i = \{m\phi_1, \ldots, m\phi_k\}$. The latest model is added to an ensemble E. An incoming point x_i is classified by its nearest $m\phi_i$. If x_i does not fall within the radius of any existing micro-classifier it is considered an outlier. New classes are identified by the labelled data in each new window and j is tuned this way. This makes the assumption that instances from a new class will be among the subset of labelled samples.

Micro-classifiers are also used in [138, 139, 82]. In each, a sliding window is used to process the stream and the contents of each new window are first classified then used to update the model. In SmScluster [138] an extension to the k-means algorithm is used to partition the partially labelled data: *K-Means with minimisation of cluster impurity* (MCI-Kmeans). The goal is minimise the intra cluster dispersion (as in traditional K-means) but also to minimise the impurity of each cluster. A cluster is considered totally pure if it contains only samples that share the same label (along with unlabelled data). Given u samples $X = \{x_1, \ldots, x_u\}$, the original K-means creates k-partitions $(X_1, \ldots, X_k)$ by minimising :

$$\sum_{i=1}^{k} \sum_{x \in X_i} ||x - c_i||^2 \tag{2.6}$$

where c_i is the centre of cluster i and $||x - c_i||$ is the Euclidean distance between x and c_i. MCI-Kmeans aims to minimise :

$$\sum_{i=1}^{k} \sum_{x \in X_i} ||x - c_i||^2 + \sum_{i=1}^{k} W_i Imp_i \tag{2.7}$$

Where Imp is the impurity of cluster i and W is the weight associated with each cluster where:

$$W_i = |X_i| D_{X_i} \tag{2.8}$$

X_i is the set of points in cluster i and D_{X_i} is the average dispersion of each of these points from c_i.

As in SSEA, at each window a new model M is created (where M is composed of a set of micro-classifiers) and M is added to an ensemble E. The size of the ensemble is fixed, when this size limit is reached, the weakest component is removed from the ensemble to make room for the latest component. 'Weakest' is the component with the lowest accuracy on the labelled portion of the latest window.

Realistic Stream Clasifier (ReaSC) [139] also uses the MCI-Kmeans method. Here, a window is read and classified and retained in memory until $p\%$ of the window has been labelled. When this happens, clustering is performed but micro-clusters containing no labelled instances are also considered 'pure'. These unlabelled micro-clusters are labelled by an inductive label propagation technique. An affinity matrix is created using all micro-clusters, this can be considered a graph with micro-clusters as nodes connected by edges. The weight of each edge is a distance metric based on a Gaussian function and the labels of the unlabelled micro-clusters are determined by their neighbours. Once all micro-clusters have been labelled, they are added to the ensemble as a set of micro-classifiers. The classification method is the same as SSEA and SmScluster, and the pruning method is the same as SmScluster whereby only a fixed number of models can be retained and the weakest is removed before adding a new one.

Semi-Supervised Pool and Accuracy Based Stream Ensemble (SPASC) [82] also keeps a fixed-size ensemble. At each new window the best components of an ensemble are selected to make predictions on a point. The selection process is based on a weighting scheme whereby a classifier's weight is determined by its prediction accuracy on labelled data points. Unlabelled points do not affect the weights. Each model M consists of several micro-classifiers (identified as in SmScluster and ReaSC). With each window, summary statistics about the window are maintained (so each component of the ensemble has an associated set of statistics about the window it was created with). If the statistics on the latest window are sufficiently similar to a previous window, the classifier associated with that window is updated with the labelled data on the most recent window, otherwise a new classifier is created and the weakest one is removed.

Stream Classification Algorithm Guided By Clustering (SCARG) [178] was proposed to address the Initially Labelled Non-Stationary scenario whereby, initially training data is available but no subsequent labels are made available throughout the stream. In the first window k-means is used to create a set of clusters. Each incoming point is classified by its nearest cluster centre and then stored in a buffer. When a sufficient number of points have been stored, the clusters are deleted and their centroids added to the buffer. k-means is performed again and the process repeats in a closed loop. SCARG was shown to identify and capture virtual drift, a change in $P(x)$. If this drift leads to real drift, then it can also be captured. However, as always in k-means the number of classes must be specified and in SCARG this cannot change.

REDELLA [121](Recurring Drifts and Limited Labelled Data) creates an incre-

Table 2.3: Overview of Classification Methods with a Scarcity of Labels

Algorithm	Labelling Method	Clustering	Classification
[208] Zhu *et al.* (2007)	AL	none	Ensmeble
[127] Lindstrom *et al.* (2010)	AL	none	SVM
[127] CDBC (2013)	AL	none	SVM
[61] Fan et al. (2004)	AL	none	Tree
[209] Active Classifier (2014)	AL	none	Tree
[88] ACL-Stream (2014)	AL	k-means	Tree
[125] SSEA (2013)	SSL	k-means	Ensemble
[138] SmScluster (2008)	SSL	k-means	Micro-classifier
[139] ReaSC (2012)	SSL	k-means	Micro-classifier
[82] SPASC (2015)	SSL	k-means	Ensemble
[178] SCARG (2015)	SSL	k-means	Nearest Neighbour
[121] Redella (2010)	SSL	k-means	Tree
[122] SUN (2012)	SSL	k-means	Tree
[177] CCEM-PL (2014)	SSL	k-means	Ensemble
[143] CSL-Stream (2011)	SSL	DBSCAN	Tree

mental decision tree and at each leaf creates 'concept' clusters (these are very similar to micro-clusters) using k-means. A point is classified by the tree and stored in the node which makes the predictions. When n points are collected in a node, clustering is performed with both labelled and unlabelled data. Unlabelled data is labelled with the majority class (of the labelled points) in the cluster. Similar approaches are outlined in [122, 177]. A tree/cluster hybrid was proposed in [143]; Concurrent Semi-Supervised Learning of Data Streams (CSL-Stream). This method differs from the previous approaches in that DBSCAN is used as the clustering mechanism as opposed to k-means.

Clusters are maintained in a dynamic tree structure and a time-dampened window is used to age old data. Nodes in a tree are described by their 'density', the weight of all the instances stored in the node. These nodes are updated only when the classifier is unstable, i.e., its accuracy on recent labelled data falls below a threshold. If drift is actively detected in this way the nodes in the tree are clustered using DBSACN to propagate labels of unlabelled data and update the tree.

An ensemble of clusters and classifiers was proposed by Zhang et al [205], where labelled instances are used to train a classifier and unlabelled instances are clustered. New instances are labelled using a majority vote that included label mapping between the classifier and the clusters. The mapping takes the form of a graph based on conditional probabilities. Another semi-supervised method based on a graph structure is the K-Associated Optional Graph (KAOG) [19].

The main methods described in this section are summarised in Table 2.3.

2.4.3 Summary of Classification Methods

In stream classification, ensemble methods are more common than single model methods. Ensembles are a natural choice for non-stationary streams because new members can be added to the ensemble and under-performing ones can be removed. This approach organically addresses the stability-plasticity dilemma (the ability of a model to retain existing knowledge *or* learn new knowledge but not to be able to both equally well) and serves as a passive method to detect and react to change. However, the majority of ensemble methods assume that the non-stationary stream will be fully labelled.

Supervised ensembles outnumber Scarcity of Label (SoL) methods. Of the two approaches for dealing with a SoL, Semi-Supervised Learning (SSL) approaches are more common than Active Learning approaches. SSL approaches often employ an ensemble and, unlike fully supervised approaches, can include clusters along with classifiers as the base members. Almost all of the SSL ensemble approaches discussed use k means as the underlying clustering algorithm; typically a stream is processed in chunks and each chunk is clustered with k-means. This approach does not reflect the recent advances in stream-clustering and is a potential shortcoming of the discussed SSL methods.

2.5 One-Class Classification

Traditionally, a classifier (or an ensemble) is used to *distinguish* between two or more classes, in one-class classification we are trying to *identify* a known class. In OCC, unlike conventional machine learning algorithms, only examples from one class are available for training. The task is to define a decision boundary around the target class (or positive class) so as to accept as many target instances while rejecting instances which do not belong to the target class (the negative class, i.e. every other class). Generally a one-class classifier takes the form:

$$(m_\alpha, x_i) \geq \theta = \begin{cases} 1: & x_i \text{ is a target} \\ 0: & x_i \text{ is an outlier} \end{cases} \tag{2.9}$$

with a point x_i, a model m with parameters α and threshold $\theta \in \mathbb{R}$. If a model's confidence of a point belonging to a certain class is greater than a user supplied threshold, θ then the point is predicted as belonging to that specific class, otherwise it is predicted as not belonging to that class. It makes no assumption for it belonging to a different class, only that it does not belong to that particular class. There are

three broad families of OCC; boundary, reconstruction, and density methods.

2.5.1 Boundary Methods

Boundary methods, as the name suggests, estimate a boundary (not necessarily spherical) around the training instances. A newly arriving point x_i is considered a target object if it falls within the model's perimeter.

The Support Vector Domain Description (SVDD) [182] is based on the Support Vector Machine (SVM) [186]. It constructs a hypersphere with the smallest possible radius that can encompass the training data. This boundary is described by a set of support vectors. A percentage of samples ξ (referred to as the slack variables) can be ignored when creating the boundary in order to create a smaller and better fitting hypersphere.

The sphere is described by a centre c and radius r, with slack variables ξ. The boundary is discovered by minimising r:

$$f(r, c, \xi_i) = r^2 + C \sum \xi_i \tag{2.10}$$

where C is the trade-off between simplicity (volume of sphere) and the number of errors on the training set.

An alternative approach to the SVDD was proposed in [171]. Here, instead of a hypersphere, a hyperplane is constructed so that is maximally distant to the origin and separates the regions that contain no training samples. Experimental results suggest that both versions of the SVDD perform comparatively and in each case a Gaussian kernel works best [100]. Online versions of one-class SVM are proposed in [176, 184], these methods make a single pass of the data and at each step add a new sample to retrain the model and remove the least relevant one. In [30] an OCC for dynamic classes was proposed. It is based on the SVDD but support vectors are time-weighted, as they become older and less relevant the boundary of the sphere changes.

Boundary methods based on a Convex Hull (CH) were proposed in [34] and [62]. The CH of a data set is the minimal convex set containing all the points. Similar to the SVDD hypersphere, if a test point lies inside the hull, it is considered to belong to the target class.

2.5.2 Density Methods

Another approach to OCC is to estimate the the density of the training data. The most simple model is the Gaussian model [25], here the training data is modelled as

a Gaussian distribution and the mean μ and covariance Σ are estimated.

$$f(x_i) = (x_i - \mu)\Sigma^{-1}(x_i - \mu) \tag{2.11}$$

where x_i is the test point and the classifier's decision ($\hat{y}_i$) is:

$$\hat{y}_i = \begin{cases} 1, \ if \ f(x_i) \leq \theta & (x_i \text{ is a target}) \\ 0, \ if \ f(x_i) > \theta & (x_i \text{ is an outlier}) \end{cases} \tag{2.12}$$

where θ is a user supplied threshold. This approach makes strong assumptions that the data is uni-modal and convex. To create a more flexible model, the Gaussian model is extended to a Mixture of Gaussians (MoG) in [50]. This is a linear combination of K Gaussian Models. An even more flexible model is obtained by using the Parzen density estimator [150]. Here a Gaussian distribution is modelled on each of the training points. These density-estimation methods have been shown to work well when there is an abundance of training data but tend to perform poorly when there is not a lot of training data, especially in high-dimensional feature spaces [183].

Distance based approaches are closely related to density methods. A nearest neighbour (NN) approach to OCC is proposed in [50]. It involves finding the distance between a test point x_i and its nearest neighbour z_i in the training data, *i.e.* $z_i = NN(x_i)$. This distance is then normalised by the nearest neighbour of z_i, *i.e.* $NN(z_i)$. x_i is considered a target if:

$$\frac{d(x_i, z_i)}{d(x_i, NN(z_i))} < \theta \tag{2.13}$$

where d is the distance between the points and θ the supplied threshold. This method has been shown to work well with high dimensional data. Another approach shown to work well in high-dimensional data (and with a small amount of training data) is an OCC based on the Minimum Spanning Tree (MST) [95]. Here, an MST is created to fit the training data. The distance between a test sample x_i and the edges of the tree is used to determine if x_i is a target. Other OCC methods are based on k-means [25] and k-centres [203].

2.5.3 Reconstruction Methods

Reconstruction methods are based on an underlying model created from the training data. If a test point fits the model, then it is considered a target; otherwise it is considered an outlier. An OCC based on Pricipal Component Analysis (PCA) was

proposed in [185]. PCA finds the orthogonal transformation which captures the variance in data. The data is mapped to a subspace. To test whether a new sample x_i is a target, the reconstruction error is computed. This error is the difference between x_i and its projection in the subspace. If x_i "fits" the model, the error should be low (within a threshold θ) and it can be considered a target. A similar approach is taken with neural networks; an Autoencoder [146] is a type of network that typically consists of three layers, where the number of neurons on the input layer and output layer are the same (equal to d, the dimensionality of the data) and the middle layer consists of much fewer neurons. The original data is "encoded" into a compressed representation in the middle layer, then the data is "decoded" back to the original input. The difference between the original input x_i and its reconstructed output $\hat{x}_i$ is used to determine whether a point is a target or not. A large difference suggests the point is an outlier. Extreme Learning Machines (ELM) have been proposed as OCC following the same method in [118, 69].

2.5.4 Ensembles of One Class Classifiers

Ensembles of OCCs have become more popular in recent years and are practical in two main scenarios; 1) where the distribution of the target class is complex and multi-model and 2) as a method to transform OCC into Multi-Class Classification (MLC) with each component of the ensemble responsible for recognising one particular class (or part of a class if the distribution is multi-modal). As in all ensembles the members should display individual quality and be diverse enough to mutually complement each other. This can be achieved by using a heterogeneous ensemble [41] or a homogeneous ensemble with each member created with different training-sets [109]. Recent studies suggest that the latter performs better [37, 109, 111].

OCCClustE [110] is a homogeneous ensemble of weighted Support Vector Machines. First the data-set is clustered into k-partitions and a classifier is trained on each partition. The ensemble's prediction comes from a majority vote from each member. As would be expected, k is crucial, and the paper proposed 10 methods for identifying k, k is defined as the number of mutually complementary areas. Though, in this instance maybe density based clustering would be a better approach.

Ban and Abe [16] proposed a heterogeneous ensemble of SVDD and PCA and showed that the performance of the ensemble is comparable to that of a regular SVM but requires less training time. They did however note that the parameters are very sensitive and affect the ensemble's performance significantly. A method to automatically find these parameters was proposed in [109] where an optimisation method based on the behaviour of the fire-fly [201] is applied.

2.6 Feature Selection in Data Streams

Feature Selection (FS) aims to identify a subset of the most relevant features $\hat{f}$ from the set of all features F. Traditionally, $\hat{f}$ would be used to cluster data and all redundant features ($\{f_i : f_i \in F \text{ and } f_i \notin \hat{f}\}$) are ignored for future points. This might not be a sensible approach to non-stationary data as $\hat{f}$ is likely to change over time. A significant change could require previous clusters to be abandoned and new clusters discovered on the latest data. This would be especially true for clustering algorithms that rely on some form of distance as a similarity metric; it might not be possible to cluster two points composed of different feature subsets. For example if the number of 'important' features changes ($|\hat{f}_t| \neq |\hat{f}_{t+1}|$,) or if a previously important feature is no longer considered important ($f_i \in \hat{f}_t$ but $f_i \notin \hat{f_{t+1}}$)).

Much research has been carried out on FS and good overviews on this research are available in [120] and [4]. The majority of this research has focused on *supervised* methods whereby a feature's importance is estimated by its correlation with the class label. Features (or subsets of features) with the greatest discriminatory power between classes are selected.

Generally, FS methods can be divided into filter methods and wrapper methods. Filter methods are independent of the model and can be seen as a preprocessing step which rank features according to some criterion and the top n features are selected. Popular methods include the Fisher Score [74], Information Gain [117] and the Pearson Coefficient [20]. Wrapper methods use a model or an underlying classifier to iteratively evaluate subsets of features. An example would be the GA-SVM [83], a genetic algorithm searches for subsets of features and these potential subsets are evaluated using a traditional Support Vector Machine.

2.6.1 Unsupervised Feature Selection

Unsupervised methods are also divided into filter and wrapper methods. Unsupervised wrapper techniques use a clustering algorithm to evaluate feature subsets [116]. This method is usually computationally expensive and succumbs to what Alelyani et al. described as the "Chicken and Egg Dilemma" [4]. When attempting to cluster and feature select simultaneously, is it better to first find features and then cluster, or first cluster and then select features?

Unsupervised Filter methods are based on the intrinsic properties of the data. The most simple, yet very effective, method of unsupervised feature selection is the maximum-variance method; the average squared deviation of a feature's value from

the mean (μ). $X = \{x_1, \ldots, x_N\}$ represents N instances, where $x_i \in \mathbb{R}^d$:

$$Var(X_i) = \frac{1}{N} \sum_{j=1}^{N} (X_{ij} - \mu_i)^2 \tag{2.14}$$

A larger variance suggests the feature has a greater representative power. The variance for each feature is calculated, the features are ranked in the descending order, and top n features are selected.

Another reasonable assumption is that data from the same class are close in the decision space. Based on this assumption, features are selected by their locality preserving power, or Laplacian Score. This idea has been applied for unsupervised FS in [17].

The Lapacian Score aims to preserve the local geometric structure in data. This local structure is modelled in a nearest-neighbour graph and features which respect this graph are selected.

- A nearest-neighbour graph G is created with N nodes and an edge is created between nodes i and j if x_i and x_j are neighbours (x_i is among x_j's k nearest neighbours or vice versa).

- A weight matrix S of G models the local structure. A Radial Basis Function function with a constant t is used to weigh the edge between nodes i and j:

$$S_{ij} = \begin{cases} e^{-\|(x_i - x_j)\|^2 \frac{1}{t}} : & \text{if } i \text{ and } j \text{ are neighbours} \\ 0 : & \text{otherwise} \end{cases} \tag{2.15}$$

- The importance of a feature is considered to be the degree to which it respects G and the weight matrix S. The Lapacian Score L for feature f_r is estimated by minimising:

$$L_r = \frac{\sum_{ij} (f_{r_i} - f_{r_j})^2 S_{ij}}{variance(f_r)} \tag{2.16}$$

A good feature will have a larger S_{ij} (thus a smaller $f_{r_i} - f_{r_j}$) and should have a high variance. So, the Lapacian Score for a good feature should be small. The Lapacian Score for each feature is calculated, the features are ranked in the ascending order, and top n features are selected.

Infinite-Feature Selection [163] selects features by exploiting the convergence properties of power series of matrices. A subset of features is analogous to a path between different feature distributions. In [29] the authors propose a filter method which selects features based on their ability to preserve the original structure of

the data. Their algorithm, Multi-Cluster Feature Selection (MCFS), measures the correlation between different features using spectral analysis [92] and selects those which can most preserve the structure. Spectral clustering is performed using the top eigenvectors of the graph Lapacian. As in the Lapacian Score, a nearest neighbour graph G and weight matrix S are created. The data manifold in S is "unfolded" to a "flat" embedding of data points and features are selected using the "flat" embedding. From S, a diagonal matrix D is created whose values are column sums of S; $D_{ij} = \sum_j S_{ij}$. From these matrices, the graph Lapacian ($L = D - W$) is created and the "flat" embedding of the data can be found by solving the generalised eigen-problem: $Ly = \lambda Dy$.

The feature scores are evaluated using the resultant eigen-vectors $Y = \{Y_1, \ldots, Y_k\}$, where k is the number of clusters in the data. MCFS scores for each feature are sorted in the descending order and the top n are selected.

2.6.2 Dynamic Feature Selection

Most of the research into FS has assumed a static batch of data but recently more work has been focusing on FS in streaming data. Again, the majority of this research has been on supervised FS. The work by Katakis et al. [96] was one of the first to address the problem FS in streaming data. Here, the authors address the problem of a large, dynamic feature space. They use the example of a text stream, the feature space being all possible words. As more text arrives, new words (features) appear and the size of the known feature space grows and changes. Cumulative statistics based on the word count in each class of document are recorded. Using the chi-squared metric the top n words in each document are selected as inputs for a classifier. As a new document arrives the cumulative statistics are updated, features can be promoted or demoted from the top n and the classifier is updated with these new features. Heterogeneous Ensemble for Feature Drift (HEFT) [144] uses a Fast Correlation Based Filter [76] as a supervised filter method to select the top features in each windowed chunk of a data stream. A classifier is trained using the top features and added to an Ensemble where each classifier is trained on a different feature subset. Carvalho et al. [33] used the weights of an online classifier to estimate the importance of each feature. Interestingly, the authors found that using some of the *lowest* ranked features improved the classification accuracy. The authors reported that using 90% of the top features and 10% of the bottom features. DX-Miner [137] is a streaming classification algorithm that incorporates dynamic FS. The algorithm can use either a supervised or an unsupervised filter method. For the supervised method, the previous three windows are stored and the Information

Gain metric is used to select the top features from these recent windows. In the unsupervised case, the authors suggest that the n highest frequency features could be used but this is not discussed any further. This was extended in [192], here the authors use DX-Miner with MCFS as the filter method. An unsupervised FS method for data streams with linear time and space was proposed in [84]. Matrix sketching is used to maintain a low rank approximation of the data. At every time-step t, the top features are selected, though *all* data until time t is used for selecting the top features. The authors reported that this gave memory problems with comparative algorithms and in a dynamic stream it is perhaps better to disregard data as the stream progresses and old data is no longer relevant.

In summary, the majority of research on FS assumes a static batch of data. The majority of research into dynamic FS for data-streams assume the supervised method and is typically used for classification tasks and not suitable for clustering.

2.7 Summary

This chapter presented an overview of the main methods for analysing non-stationary data-streams. Typically, a (potentially unbounded) stream is processed in windows; fixed sized 'chunks' of data or time-weighted data. This windowing technique is prevalent in stream clustering (Section 2.3), classification (Section 2.4), and feature selection (Section 2.6). Processing the data this way allows practitioners to apply traditional, batch methods to continuous data-streams where time and memory constraints would otherwise render them unsuitable. An example of this is the common online/offline approach to stream-clustering where data is first summarised online and then clustered in batches off-line using traditional methods like k-means or DBSCAN.

This approach is also prevalent in dynamic classification where a traditional online learner is incrementally trained on the latest window of the incoming stream. In non-stationary streams a change detection mechanism is often employed alongside the classifier. This mechanism detects change using statistical measures or the classifier's performance as compared with the ground truth. These methods are active in that they continuously examine the stream for significant deviations which signal change. If change is detected the classifier is discarded and a new model is trained on the latest data. Other methods use a passive approach to change. Here, change is assumed and the model is continuously updated with the latest data. This is a common approach with ensemble methods. Ensembles can passively deal with change by adding a new member (trained on the most recent data) and removing

the weakest member (tested on the most recent data). Ensembles with a fixed sized sliding window are very common in both the supervised framework and in a stream with a scarcity of labels.

In a stream with a scarcity a labels, ensembles often contain clusters alongside classifiers. This mixed ensemble has been applied with an active learning approach but is more common in the semi-supervised framework. A typical approach is to process the stream in windows and cluster each window. Discovered clusters will contain both labelled and unlabelled data, and labels are propagated to all points based on these clusters. Traditional supervised techniques are then implemented after all points have been labelled. Almost all of these ensemble approaches use k-means as the underlying clustering algorithm. There is a disparity here with the stream-clustering literature; almost none of the recent stream-clustering algorithms use k-means ; the main reasons are that it is an iterative process (not single pass), identified clusters are spherical (not as robust to noise) and k must be specified (unsuitable if there is potential for concept evolution). Recent advances in stream clustering include online methods which are unaffected by the aforementioned limitations but these have not been applied to non-stationary mixed ensembles.

Feature selection methods for data streams were discussed and, as in stream-classification, supervised methods outnumber unsupervised methods.

Chapter 3

Stream Clustering with Ants

This chapter introduces Ant Colony Stream Clustering. Although there is a large body of research on static batch clustering using ants few are designed to cope with data streams and none extend the 'pick-and-drop' model prevalent in batch clustering (see Section 2.1.6).

Density-based clustering identifies clusters as areas of high density in the feature space separated by areas of low density. It is an attractive approach to dynamic stream clustering for three reasons: 1) the number of clusters does not need to be specified. This is useful in the presence of concept evolution where the number of natural clusters will change. 2) Arbitrary shaped clusters (not only spherical) can be discovered. This is useful in the presence of noise/outliers and 3) the micro-cluster acts as both the clustering mechanism and as a natural summarisation method because a number of similar, local points can be represented by a single micro-cluster.

A density approach can address the problem of a shifting number of non-stationary clusters and provides a mechanism to summarise these clusters. The method outlined in this chapter combines density methods with artificial ants to speed-up and improve clustering performance.

Ant-Colony Stream Clustering (ACSC) identifies clusters as 'nests' of micro-clusters in dense areas of the data. The potential advantage of using an ant colony is that the exhaustive search for each point's appropriate micro-cluster, and subsequently the nearest neighbour of each micro-cluster, is replaced with stochastic sampling from nests. Nests are first created by Merging ants and then refined by Sorting ants (using the Pick-and-Drop model).

In other stream-clustering algorithms which use micro-clusters; (DenStream [31], FlockStream [66], etc.), micro-clusters are controlled by two parameters; ϵ which specifies the maximum radius and $minPoints$ which specifies the minimum number

of points within ϵ for the micro-cluster to be defined as 'dense'. In ACSC, each point is initially considered as its own micro-cluster and so the *minPoint* parameter is effectively 1 and therefore not required. This also removes the complication of defining micro-clusters as either *core*, *potential* or *outlier* (as in other density algorithms) because each micro-cluster is considered equal. To further simplify, the concepts *density-reachable*, *directly density-reachable*, and *density-connected* are merged into one concept; *density-reachable*. This determines if two micro-clusters are connected and therefore part of the same cluster. ACSC assigns points to a cluster *before* merging micro-clusters so when a new micro-cluster is directly density-reachable to any in the cluster, it is density-reachable and density-connected to all micro-clusters in the cluster. This simplifies the process, reduces the overall complexity and allows for effective sampling.

The main contributions of this chapter can be summarised as:

- A contribution to the stream-clustering literature where currently there is an imbalance between ant-based static clustering and ant-based stream clustering.
- A novel algorithm based on the observed sorting behaviour of ants. Experimental results show that the algorithm is scalable, robust to noise and favourable to leading ant-clustering and stream-clustering algorithms.
- The algorithm takes fewer parameters than its peers and requires considerably less computational time.

The work presented in this chapter has been previously published in [55] and [56].

3.1 Ant Colony Stream Clustering

ACSC reads the stream in windows; at each iteration a fixed-sized, non-overlapping section of the stream is considered. The solution provided by ACSC is a set of clusters containing density-reachable micro-clusters. ACSC works in two steps: 1) rough nests are identified in a single-pass of the window and 2) these rough nests are refined. The nests established at the end of step 2 are considered to be the final clusters and their summary statistics are stored off-line.

3.1.1 Preliminaries

To recap, a micro-cluster is an extension to the Current Feature Vector [204] and is controlled by an additional *global* value ϵ which determines the micro-cluster's

Algorithm 7 Merge Operation

Input : ϵ-neighbourhood, 2 micro-clusters; a and b
Output : Merged micro-cluster *iff* operation successful

```
1: Create new, empty micro-cluster c
2: Initialise c := a
3: Add b to c (Eq. (3.3))
4: r := radius of c (Eq. (3.2))
5: if (r ≤ ε) then merge successful
6:     Delete a and b
7:     Return c
8: else
9:     Delete c
10:    Return false
11: end if
```

maximum radius. So, a micro-cluster containing N points $\{\vec{X}_i\}$, $i = \{1, ..., N\}$, is described using three components: the number of data points described by the micro-cluster (N), the Linear Sum (LS) of each dimension (i.e.,$\sum_{i=1}^{N} \vec{X}_i$), the Squared Sum (SS) of each dimension (i.e., $\sum_{i=1}^{N} \vec{X}_i^2$). LS and SS are d-dimensional arrays, where d is the number of dimensions in a point. From these, the radius r and centre c of the micro-cluster can be determined [1]:

$$c = \frac{LS}{N} \tag{3.1}$$

$$r = \sqrt{\frac{SS}{N} - \left(\frac{LS}{N}\right)^2} \tag{3.2}$$

A micro-cluster $mc_i = [N, LS, SS]$ and mc_i can absorb point p_i if, after updating the LS and SS of mc_i with p_i, $radius(mc_i) \leq \epsilon$. Similarly, two micro-clusters mc_i and mc_j can attempt to merge into a single micro-cluster mc_k as follows:

$$mc_k = (N_i + N_j, LS_i + LS_j, SS_i + SS_j) \tag{3.3}$$

If $radius(mc_k) \leq \epsilon$, the clusters merge; otherwise, the merging operation fails. The pseudo-code for this process is outlined in Algorithm 7. Micro-clusters m_i and m_j are said *density reachable* if:

$$dist(c_{m_i}, c_{m_j}) \leq \epsilon, \tag{3.4}$$

where c_{m_i} and c_{m_j} are the centres of micro-clusters m_i and m_j, respectively.

Algorithm 8 Find Clusters

Input : Window
Output : Clusters, Cluster Similarity

```
1: while <Window> do
2:     for <each data point> do
3:         if <clusters> then
4:             Find best cluster (Eq. (3.5))
5:             Add point to cluster
6:             Update cluster similarity (Eq. (3.6))
7:         else if <No clusters ||
8:               No suitable cluster> then
9:             Create new cluster
10:            Add point to cluster
11:            Update cluster similarity (Eq. (3.6))
12:        end if
13:    end for
14: end while
15: return Clusters, Cluster Similarity
```

3.1.2 Creating Initial Nests

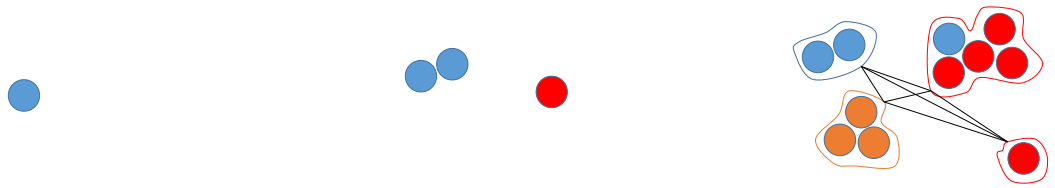
The steps begins by reading the latest window in the stream, initially there are $windowSize$ number of points. In the biological metaphor, these points are Merging ants and a nest is a grouping of similar Merging ants. Merging ants are iteratively assigned to nests representing dense areas of the data. The first ant creates the first nest and subsequent ants can either join an existing nest or form a new one.

Each ant visits each nest in succession and evaluates the nest's suitability by comparing itself with all ants currently in the nest. Formally, the similarity of ant a with nest k is defined as follows:

$$Sim(a,k) = \frac{1}{n_k}\sum_{j=1}^{n_k} dist(a,k_j), \tag{3.5}$$

where nest k already has n_k ants present in it (i.e., $k = \{k_1, k_2, ..., k_{n_k}\}$). Ant a joins the most similar nest *provided* its similarity score is equal to or below ϵ. If not, it forms a new nest. The parameter ϵ determines the maximum radius for a micro-cluster in the subsequent step and also serves as the minimum suitability measure in this step.

As an ant evaluates each nest, it 'remembers' its similarity with each nest (Eq. (3.5)). Upon joining a nest or establishing a new nest, the similarity of the selected nest with all its neighbouring nests is updated. This similarity update happens in a decentralised, iterative way; similar to pheromone trails in an ant colony

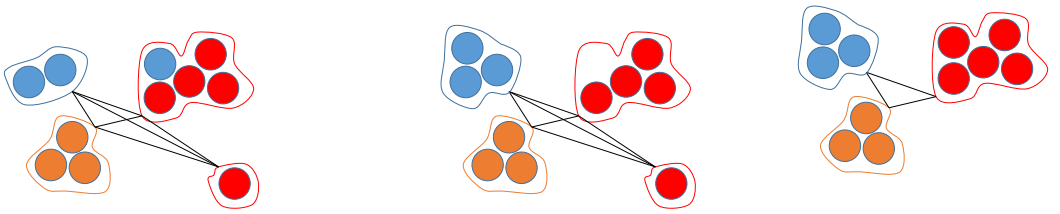


a) First ant creates the first nest

b) Subsequent ants either join existing nest or create a new nest

c) At the end of this step, rough clusters are formed and their similarity recorded

Figure 3.1: **Create Initial Nests**



c) Nests from step one are sorted

d) Rough nests are purified

e) Smaller nests dissolve and their contents move to larger, similar nests

Figure 3.2: **Sort Nests**

and these similarity scores are recorded in a matrix, which is referred to as the Pheromone Matrix (PM). The pheromone trail to each neighbouring nest is a rolling average updated whenever a new ant joins the nest. Formally, the pheromone trail between nests k and l is the average of each ant i in nest k's similarity (Eq. (3.5)) with nest l:

$$ph(k,l) = \frac{1}{n_k}\sum_{i=1}^{n_k} Sim(k_i, l), \tag{3.6}$$

where n_k is the number of ants in nest k and k_i is the i-th ant in nest k.

At the end of this step there are j nests ($\{k_1, \ldots, k_j\}$), each containing n_k ants and a PM describing the similarity between each pair of nests as follows:

$$\mathbf{PM} = \begin{array}{c} \begin{matrix} k_1 & k_2 & \ldots & k_j \end{matrix} \\ \begin{bmatrix} 0 & ph(k_1,k_2) & \ldots & ph(k_1,k_j) \\ ph(k_2,k_1) & 0 & \ldots & \ldots \\ \vdots & \vdots & \ddots & \vdots \\ ph(k_j,k_1) & \ldots & \ldots & 0 \end{bmatrix} \end{array} \begin{matrix} k_1 \\ k_2 \\ \ldots \\ k_j \end{matrix} \tag{3.7}$$

This step is outlined in Algorithm 8. An illustrative example of the process is displayed in Figure 3.1. Here, the different colours signify different concepts. Each ant represents a micro-cluster, the first ant forms the first nest, subsequent ants either join an existing nest or, if too dissimilar, form a new nest. As nests form, each nest's similarity with all of the other nests is recorded in the form of pheromone trails between each - represented by the black edges.

3.1.3 Sorting Nests

The previous step creates initial nests in a single-pass of the window. The nests identified in the first step are often rough, impure and too-many. In this step, micro-clusters are created, merged, and inter-cluster sorting is performed.

Initially, each d-dimensional point p in each cluster is treated as its own micro-cluster m. This micro-cluster will have a radius of 0 and a centre of p. Formally, we have:

$$\begin{aligned} m.N &= 1 \\ m.LS_i = p_i, i &= \{1, \cdots, d\} \\ m.SS_i = {p_i}^2, i &= \{1, \cdots, d\} \end{aligned} \tag{3.8}$$

where p_i is the i^{th} dimension of point p.

Before sorting begins, each micro-cluster attempts to merge with other micro-clusters in the same cluster. The merging operation is performed by comparing each micro-cluster with every other in the same cluster (Eq. (3.3), Algorithm 8). Merging at this step has two advantages:

- Only neighbouring micro-clusters attempt to merge. This prevents the unnecessary computation of comparing micro-clusters in different dense areas.
- During the sorting phase, n points represented by a micro-cluster can be moved in a single operation. This speeds up the sorting process and reduces the number of pairwise comparisons

Merging ants were used in the previous step to create the initial nests. These are no longer used and, in this phase, 'Sorting ants' are created. A Sorting ant is assigned to each cluster, so each Sorting ant is native to its own cluster. Sorting ants probabilistically decide to pick-up a micro-cluster from their cluster. A micro-cluster mc_i is chosen at random from cluster k and is iteratively compared with $nSamples$ micro-clusters in the same cluster. The Euclidean distance from the centre of mc_i to each of the selected micro-clusters is calculated and if both are density-reachable (Eq. (4)), then a *reachable* count is incremented. The probability of a pick-up is

Algorithm 9 Sort Clusters

Input : Initial Clusters, Cluster Similarity
Output : Sorted Clusters

```
1: Create micro-clusters (Eq. 3.8)
2: Merge micro-clusters in each cluster (Algorithm 7)
3: Assign Sorting Ant to each cluster
4: while <!Stop Condition> do
5:     for <each ant> do
6:         if <!Sleeping> then
7:             Probabilistically pick-up (Eq. (3.9))
8:             if <Carrying> then
9:                 Move to most similar cluster
10:                Probabilistically drop (Eq. (3.9))
11:            end if
12:        end if
13:        Update similarity information (Eq. (3.6))
14:        Update sleepCounter
15:    end for
16: end while
17: return Clusters
```

calculated as follows:

$$P_{pick} = 1 - \frac{reachable}{nSamples} \tag{3.9}$$

It is important to note here that if the number of micro-clusters n in cluster c is fewer than $nSamples$, then only n comparisons are made. However, P_{pick} is still calculated using $nSamples$. This ensures a higher probability of a pick-up in clusters containing fewer micro-clusters. This leads to the dissolution of smaller clusters and their incorporation into larger, similar clusters.

If a micro-cluster is successfully picked-up, the Boolean variable *carrying* is true and the Sorting ant moves to a neighbouring cluster and attempts to drop it.

Sorting ants move to the most similar cluster (using the PM created in the previous step) ensuring that they do not attempt to drop micro-clusters in clusters that are not similar to their own. A Sorting ant attempts to drop its micro-cluster in the new cluster based on the inverse of Eq. (3.9). If the dropping operation is successful, the micro-cluster is moved to the new cluster; otherwise, the micro-cluster remains in its original cluster. The ant returns to its native cluster and updates the PM between the two nests with the latest similarity score, see Eq. (3.5). An illustrative example of the sorting step is displayed in Figure 3.2. Here, the rough nests created in the first step are refined.

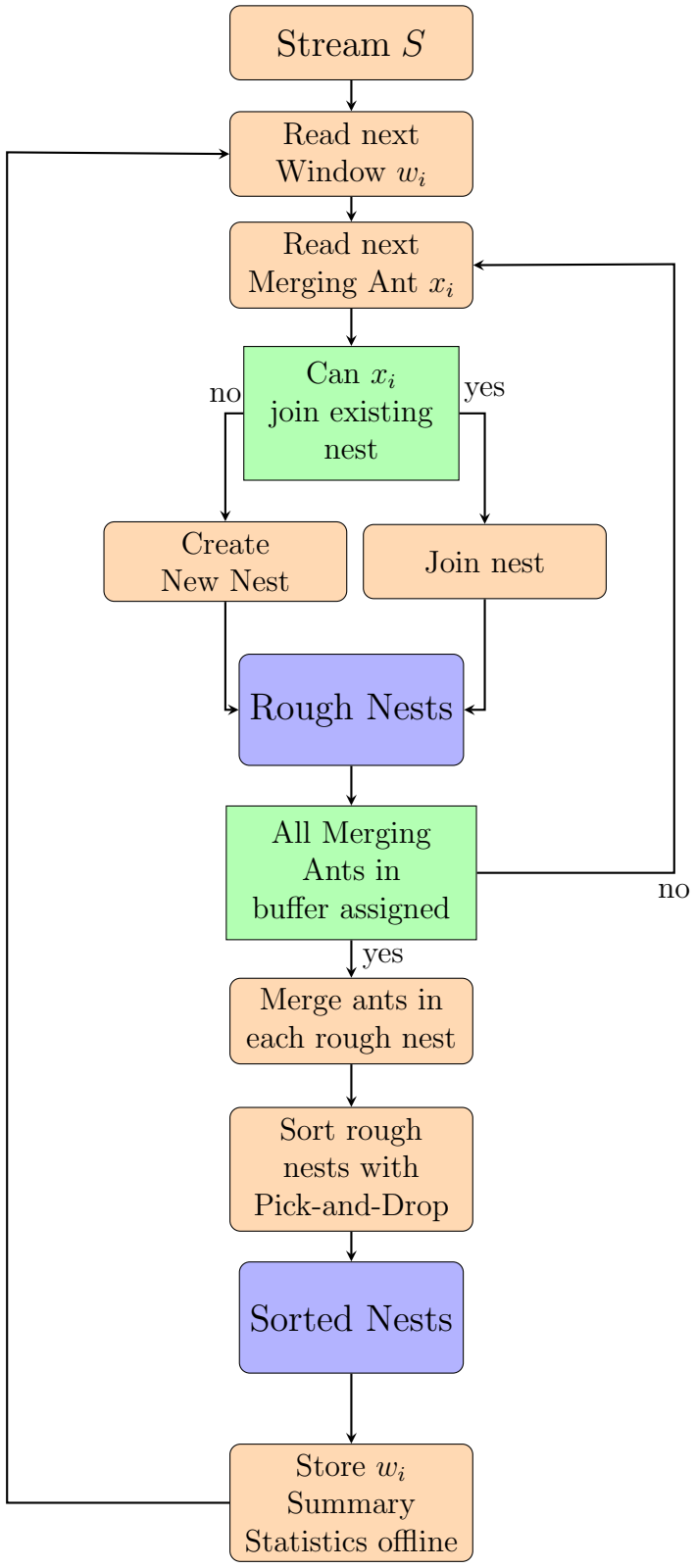
Each Sorting ant continues to attempt sorting until either the cluster is empty

(all of its contents have been moved to another cluster) or the Sorting ant is 'asleep'. Each ant has a counter and if a pick-and-drop operation is unsuccessful, either picking or dropping, this counter is incremented. When the counter reaches $sleepMax$, then the cluster is considered to be sorted and a Boolean counter *sleeping* is true. The counter is reset to zero after a successful operation or if a new micro-cluster is placed in the cluster by a foreign Sorting ant. When all ants are sleeping, the stop condition is met.

This step purifies each cluster and causes many smaller, similar clusters to dissolve and form one larger cluster. Clusters containing only one micro-cluster are considered to be outliers and the clustering solution is given as the set of non-empty clusters. Each cluster contains a grouping of density-reachable micro-clusters which summarize the partitioned areas of high-density in the feature space. These summary statistics are stored offline and the next window in the data stream is processed. This step is outlined in Algorithm 9.

The entire process is presented as a flowchart in Figure 3.3.

Figure 3.3: ACSC Flowchart



3.2 Experimental Study

The performance of ACSC is evaluated on stationary batch data and non-stationary streams across three metrics. Because there are so few ant-based stream-clustering algorithms, ACSC is compared with four popular *static* ant clustering algorithms. ACA [187] extends the original pick-and-drop implementation [132] by introducing a cooling scheme for the picking probabilities. ACAm [78] extends this by associating a short term memory with each ant. The heuristic is further improved in ATTA [77] by using a colony of heterogeneous ants. Each of these algorithms extend the pick-and-drop model outlined in Algorithm 1 in Section 2.1.6. *AntClust* does not use the pick-and-drop model for clustering but is instead inspired by the chemical recognition system of ants. The performance of these algorithms is taken from results already published in the literature.

ACSC is subsequently compared with three leading stream clustering algorithms on non-stationary streams; DenStream [31], CluStream [1] and ClusTree [112]. Each of these peer algorithms are evaluated using the Massive Online Analysis (MOA) [24] open source software.

3.2.1 Performance Metrics

ACSC is evaluated across three metrics: Purity, F-Measure [91] and the Rand Index [157]. Each dataset is labelled and the ideal "correct" clustering solution is known, so performance is measured with respect to this ground truth. In each metric, a bad clustering will have a value close to 0 and an ideal clustering solution will have a value of 1.

The Purity metric measures how homogeneous a cluster is. A cluster is assigned to the class which appears most frequently within the cluster, the accuracy of this is evaluated by summing the instances of this class and dividing by the total number of instances in the cluster. The *F-Measure* (sometimes called *F-Score* or *F1-Score*) is the harmonic mean of the precision and recall scores obtained by the algorithm.

In the following, R represents the clustering result returned by the algorithm. R contains n clusters. In every identified cluster R_i ($i = \{1, \cdots, n\}$), V^i represents the most frequently appearing class label in cluster R_i, V^i_{sum} is the number of instances of V^i in R_i, and V^i_{total} represents the total number of instances of V^i in the current window. From these, we define the following features for cluster R_i:

$$precision_{R_i} = \frac{V^i_{sum}}{|R_i|} \tag{3.10}$$

$$recall_{R_i} = \frac{V^i_{sum}}{V^i_{total}} \tag{3.11}$$

$$Score_{R_i} = 2 * \frac{precision_{R_i} * recall_{R_i}}{precision_{R_i} + recall_{R_i}} \tag{3.12}$$

Overall, Purity (P) and F-Measure (F) can now be expressed in terms of the total number of clusters identified, as follows:

$$P = \frac{1}{n} \sum_{i=1}^{n} precision_{R_i} \tag{3.13}$$

$$F = \frac{1}{n} \sum_{i=1}^{n} Score_{R_i} \tag{3.14}$$

The Rand Index (R) is a measure of agreement between two clustering solutions; the solution identified by the algorithm and the *ideal* clustering solution known from the ground truth. It measures the number of decisions that are correct by penalising false negatives and false positives, as follows:

$$R = \frac{TP + TN}{TP + FP + TN + FN}, \tag{3.15}$$

where TP, TN, FP, and FP denote the number of true positive, true negative, false positive and false negative decisions, respectively.

The performance of ACSC (stochastic) is compared with results already published in the literature and also with three *deterministic* streaming algorithms (DenStream, CluStream and ClusTree). To statistically analyse the results on the above metrics, the non-parametric One-Sample Wilcoxon Signed-Rank Test is used [196]. The null hypothesis that the distribution of the ACSC results are symmetric around the corresponding peer result is rejected with $p < 0.05$.

3.2.2 Datasets

ACSC is compared with other ant-based clustering solutions across three well known and popular non-stationary datasets; Iris, Wine, and Zoo. These datasets were taken from the UCI Machine Learning Depository[1] and the details of each are presented in Table 3.1. These datasets are originally sorted by class so to remove any potential bias, each dataset is randomly shuffled.

To evaluate the performance of ACSC over non-stationary streams, four datasets were used. Two datasets are synthetic and are taken from the non-stationary dataset

[1]http://archive.ics.uci.edu/ml/

Table 3.1: Description of Data

		Classes	Features	Examples	Drift Interval	Type
Non-Stationary						
	1CDT	2	2	16,000	400	Synthetic
	4CR	4	2	144,400	400	Synthetic
	Network Intrusion	5	42	494,000	unknown	Real
	Forest Cover	7	54	580,000	unknown	Real
Stationary						
	Iris	3	4	150	none	Real
	Wine	3	13	178	none	Real
	Zoo	7	17	101	none	Real

Table 3.2: Comparative Performance on Static Data

	AntClust			*ATTA*			*ACA*			*ACAm*			*ACSC*		
	P	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*
Iris	0.89	0.84	0.84	–	0.81	–	0.77	0.77	0.78	0.81	0.81	0.81	**0.92**(s+)	**0.90**(s+)	**0.89**(s+)
Wine	0.94	0.73	0.73	–	0.88	–	0.86	0.86	0.83	0.86	0.87	0.85	**0.94**(s+)	**0.90**(s+)	**0.88**(s+)
Zoo	0.66	0.68	**0.90**	–	0.81	–	0.77	0.76	0.88	0.76	0.77	0.89	**0.97**(s+)	**0.85**(s+)	0.88(s-)
Average	0.83	0.75	0.82	–	0.83	–	0.80	0.80	0.83	0.81	0.82	0.85	**0.94**(s+)	**0.90**(s+)	**0.88**(s+)

used in [178] and made publicly available by the authors[2]. The first synthetic dataset $1CDT$ consists of one static class and one non-stationary class and represents virtual concept drift, a change in $P(x)$, the second dataset $4CR$ consists of four classes. Each class rotates anti-clockwise and moves into positions previously occupied by a different class. This represents real drift, a change in $P(y|x)$.

ACSC is also tested on two real data-streams: the Network Intrusion benchmark dataset[3] used in [66] and [31] and the Forest Cover-Type data-set[4]. The Network Intrusion data-stream is composed of seven weeks of simulated network requests on the DARPA network. Requests can be "normal" or "malicious". There are four non-stationary "malicious" classes and these four malicious classes exhibit substantial drift as they are composed of twenty three different types of attack.

The Forest Cover data-stream is composed of 54 cartographic variables describing forest coverage in Roosevelt National Forest of northern Colorado and is widely used in the stream-mining literature [1, 112, 71]. The full details of each dataset are presented in Table 3.1. Each of these datasets display characteristics of concept drift and concept evolution. These datasets have been transformed into a stream by taking the input order as the streaming order.

[2]https://sites.google.com/site/nonstationaryarchive/
[3]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[4]https://archive.ics.uci.edu/ml/datasets/covertype

Table 3.3: Comparative Performance on Non-Stationary Streams

	DenStream			*CluStream*			*ClusTree*			*ACSC*		
	P	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*
1CDT	0.99	0.82	0.77	**1.0**	0.88	0.80	**1.0**	0.89	0.82	0.99(s-)	**0.99**(s+)	**0.99**(s+)
4CR	**1.00**	0.67	0.71	**1.00**	0.89	0.89	**1.00**	0.89	0.89	0.99(s-)	**0.95**(s+)	**0.97**(s+)
Network	**1.00**	0.80	0.81	0.35	0.13	0.36	0.36	0.16	0.3	**1.0**(=)	**0.95**(s+)	**0.95**(s+)
CForestCover	**0.89**	0.10	0.51	0	0	0	0	0	0	0.88(s-)	**0.59**(s+)	**0.64**(s+)
Average	0.88	0.50	0.64	0.59	0.49	0.58	0.59	0.51	0.58	**0.93**	**0.77**	**0.83**

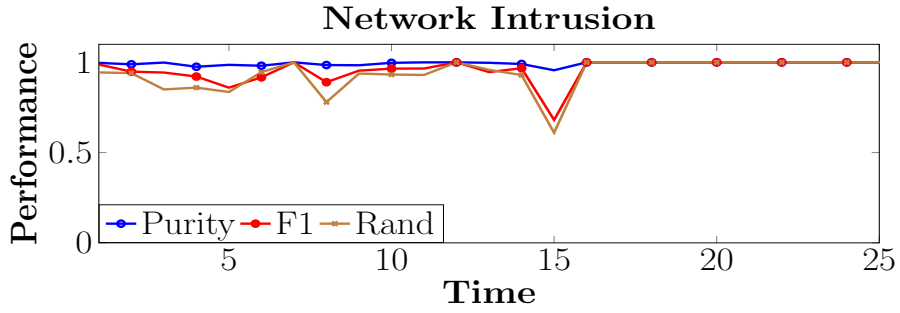


Figure 3.4: Progression of the Network-Intrusion Stream

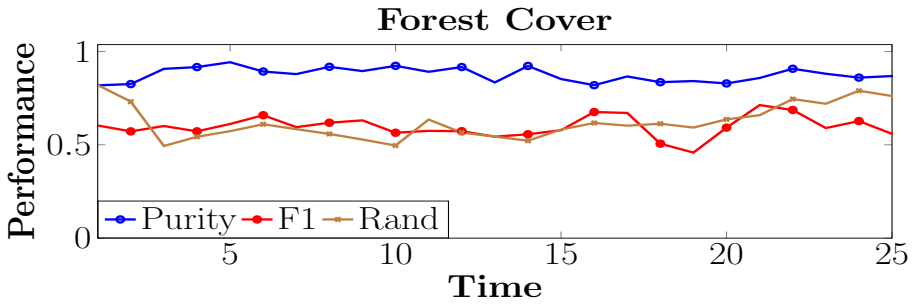


Figure 3.5: Progression of the Forest-Cover Stream

3.2.3 Clustering Quality Evaluation

To evaluate ACSC on the static datasets the algorithm was tested on one window with *windowSize* set to the number of samples. ACSC is compared with four static ant clustering algorithms and the performances of these algorithms were taken from results already published in the literature. The Purity and Rand Index for the ATTA algorithm are omitted as they are not available in the literature. The comparative results are presented in Table 3.2. The average of 30 runs of ACSC is presented along with result of the Wilcoxon Signed-Rank Test, where "$s+$" indicates ACSC performs significantly better than the best peer result and "$s-$" indicates ACSC performs significantly worse. ACSC performs significantly better on each metric in the Iris and Wine datasets and is outperformed only by *AntClust* on the Rand Index measure on the Zoo dataset. The overall naverage shows that ACSC outperforms the

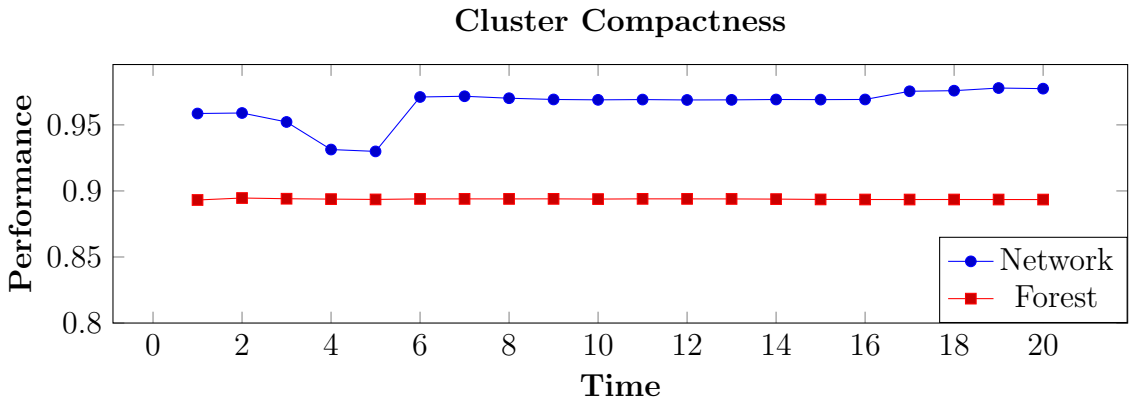


Figure 3.6: Cluster Cohesion.

others on these three datasets on all metrics. To evaluate ACSC on non-stationary streams, ACSC is compared with DenStream, CluStream and CluStream. Table 3.3 displays a comparative evaluation of each algorithm across the entire stream. The peer algorithms are deterministic but ACSC is stochastic so the displayed results are the average, along with the Wilcoxon test over 30 runs. ACSC achieves the best Rand Index and F1 scores on each dataset and on average, is the best overall. On the final stream; Forest Cover, the reported results are obtained using the full dataset (containing 54 features, continuous and discreet). CluStream and ClusTree were unable to find a clustering solution on this full dataset. Previous studies report using a subset of the data (the first 10 continuous variables). While Table 3.3 shows the mean values across the whole stream, the on-line performance of ACSC as a stream progresses is displayed in Figs. 3.4 and 3.5. The two real data-streams (Network Intrusion and Forest Cover) are displayed and the algorithm's performance over the first 25 windows (size 1,000) are displayed.

3.2.4 Internal Evaluation of Discovered Clusters

In the previous sections, discovered clusters were evaluated against the known ground truth (external evaluation). In this section we evaluate the clustering performance using an internal metric: cluster compactness (or cluster cohesion). Cohesion is a measure of how similar a data instance is to its own cluster. For each instance i, A denotes the cluster to which i belongs. The cohesion ($c(i)$) of i to all other instances in A:

$$c(i) = \frac{1}{\|A\| - 1} \sum_{j \in A, j \neq i} dist(i, j) \tag{3.16}$$

The final score is the average cohesion of all discovered clusters.

Both real streams (Forest Cover and Network Intrusion) were evaluated with

this metric. For each stream, the first 20 windows (with window size equal to 1,000) were considered. At each time step the cohesion of the clusters were plotted and the results are displayed in Figure 3.6. It can be seen that on each stream the algorithm discovers compact clusters at each window. On the Forest Cover stream, the cluster compactness is stable and consistent, though there is some fluctuation on the Network Intrusion stream.

3.3 Effect of Merging Ants and Sample-Size

In this section, the effect of the first step of the algorithm is illustrated. In the first step, the algorithm makes a single pass of a window, incrementally forming clusters using Merging ants. A point p's similarity with an existing cluster C is evaluated using the Euclidean distance from p with a sample (without replacement) from nest k. The $nComp$ parameter determines the size of this sample:

$$nSamples = WindowSize * nComp. \tag{3.17}$$

A smaller value for $nComp$, say 0.05, will result in a smaller sample taken from the nest, fewer comparisons made and, intuitively, a faster run. A larger value for $nComp$ will require more comparisons, slowing the algorithm but offering, potentially, greater accuracy with less variance in results. A value of 1.0 for $nComp$ requires a comparison with each ant in every nest (not just a sample) effectively making this phase of the algorithm deterministic.

Table 3.4 displays the performance (over 10 runs) of ACSC on the Network Intrusion data-stream with gradually increasing values for $nComp$. For simplicity the performance of the algorithm is taken as the average of the Purity, F1-Measure and Rand Index metrics (along with the standard deviation). The performance is presented alongside the running time (in seconds) on windows of varying size; from 1,000 to 5,000 samples per window.

It can be seen that the performance of the algorithm improves only very slightly with an increase in $nComp$ whereas the running time increases with larger values. For example, with each window size, a value of 1.0 takes over twice as long as a value of 0.1 with only a minimal improvement in performance. For all of the results reported in the previous tables, $nComp$ is set to 0.1.

Table 3.4: Effect of the nComp parameter on Speed and Performance

Ncomp.	$Win. = 1,000$		$Win = 2,000$		$Win = 5,000$	
	$Perform.$	$RunTime$	$Perform.$	$RunTime$	$Perform.$	$RunTime$
0.05	0.955 (0.2)	15.3 (0.517)	0.951 (0.81)	35.5 (13.5)	0.943 (1.8)	115.4 (4.30)
0.1	0.961 (0.02)	17.9 (0.565)	0.957 (0.02)	36.02 (0.42)	0.946 (0.06)	121.35 (8.20)
0.2	0.965 (0.01)	22.2 (0.168)	0.957 (0.01)	45.07 (0.83)	0.946 (0.01)	125.5 (2.79)
0.4	0.967 (0)	30.2 (0.183)	0.958 (0.01)	59.3 (0.19)	0.946 (0)	145.61 (5.71)
0.8	0.968 (0)	32.9 (0.140)	0.958 (0)	71.8 (1.2)	0.947 (0)	217.01 (6.1)
1.0	0.968 (0)	38.8 (0.031)	0.958 (0)	76.0 (0.7)	0.947 (0)	247.31 (1.19)

Table 3.5: Effect of Sorting Ants on Wine Data

Before

Nest	class 1	class 2	class 3
1	[59	9	0]
2	[0	51	6]
3	[0	8	0]
4	[0	1	0]
5	[0	2	0]
6	[0	0	40]
7	[0	0	2]

After

Nest	class 1	class 2	class 3
1	[59	9	0]
2	[0	62	6]
3	[0	0	0]
4	[0	0	0]
5	[0	0	0]
6	[0	0	42]
7	[0	0	0]

Table 3.6: Effect of Sorting Ants on Network Intrusion Stream

Before

Nest	class 1	class 2
1	[841	0]
2	[48	0]
3	[31	2]
4	[78	0]

After

Nest	class 1	class 2
1	[857	0]
2	[70	0]
3	[1	2]
4	[70	0]

3.4 Effect of Sorting Ants

In the first phase of ACSC, rough clusters are formed using Merging ants. In the second phase, Sorting ants are assigned to each cluster and inter-cluster sorting is performed. The effect of the Sorting ants on the initial clusters is reported.

The stopping condition for this phase is determined by the $sleepMax$ parameter; the number of unsuccessful sorting attempts allowed for each ant before it is "asleep". A $sleepMax$ of 0 means that this phase is never performed.

Two datasets are used to illustrate this phase; the Wine dataset and a window

from the Network Intrusion stream. The Wine dataset contains three classes with a distribution of [59, 71, 48]. While the window (size = 1,000) from the Network Intrusion stream has a distribution of [998, 2]. Table 3.5 shows the clusters identified in the Wine dataset. The three natural clusters are initially grouped into 7 clusters with an overall purity = 0.97, F-Score = 0.86 and Rand Index = 0.85. After the Sorting phase (with a $sleepMax$ of 3) three clusters are identified with an overall purity = 0.94, F-Score = 0.91 and Rand Index = 0.90.

Table 3.6 shows the Network Intrusion window before and after Sorting ants, with purity = 0.98, F1 = 0.51, and Rand Index = 0.72. The sorted clusters have a similar purity but higher F1 and Rand Index score (0.86 and 0.75, respectively). Also, the cluster (cluster 3) which describes class 2 is better represented.

Imitating their biological counterparts, the Sorting ants are biased to picking-up isolated items and dropping them in denser areas. The final clusters are a closer representation of the true underlying structure. The purity score is lower than the initial clusters as the *average* purity is measured. For example, cluster 4 in Table 3.5 contains a single micro cluster and so has 100% purity and the overall average increases. However, these sparse clusters lower the F-Score and Rand Index metrics. Taken on its own, Purity can be a misleading metric as it does not consider the actual topology of the data. A similar performance can be seen in the Network Intrusion dataset (first 100 windows, with a window size of 1,000).

3.5 Complexity Analysis

To evaluate the time requirement of the ACSC, speed is measured in the amount of time (seconds) it takes to process the entire Forest Cover stream and also the average time it requires to process a single window (size 1,000). For comparison, the time requirements of the peer algorithms are also evaluated and the results are reported in Table 3.7.

Table 3.7: Total time required (seconds) to process entire Forest Cover stream and average window processing time using window size of 1,000. * Faster algorithms did not discover a clustering solution

	DenStream		*CluStream*		*ClusTree*		*ACSC*	
	Total,	Window	Total,	Window	Total,	Window	Total,	Window
1*CDT*	05.74	0.38(0.06)	01.69	0.11(0.02)	01.22	0.07(0.01)	**0.71**(0.01)	**0.05**(0.02)
4*CR*	50.62	0.29(0.04)	11.78	0.09(0.01)	12.11	0.09(0.01)	**09.28**(0.1)	**0.06**(0.01)
Network	94.41	0.19(0.77)	106.21	0.22(0.18)	22.11	0.06(0.3)	**20.63**(0.3)	**0.04**(0.02)
ForestCover	278.5	0.56(0.09)	26.62	0.04(0.02)*	22.07	0.03(0.02)*	**49.53**(1.07)	**0.08**(0.02)

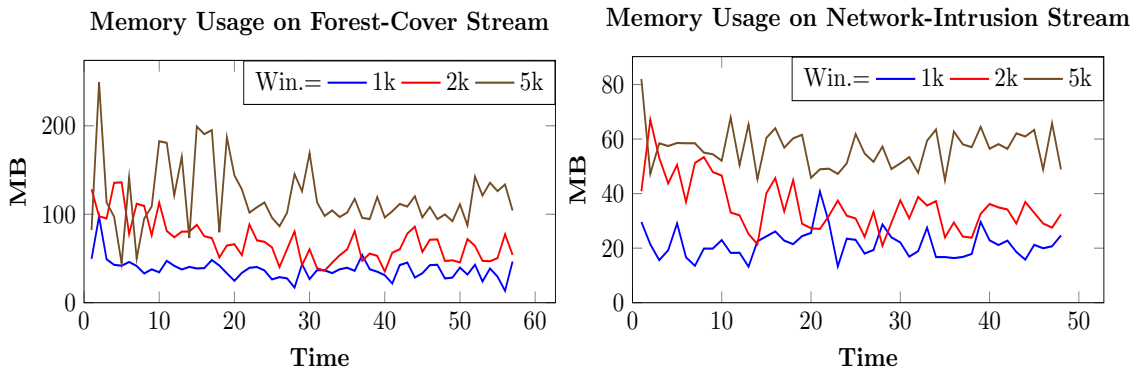


Figure 3.7: Memory Requirements

The memory requirement of the algorithm is a function of the window size. The window is read in a single pass, summarized into a smaller number of micro-clusters and then deleted. These micro-clusters are operated on and stored. So the overall memory usage is determined by the size of the window (the number and dimensionality of points) loaded into memory. A commercial profiler [90] is used to accurately measure the memory usage of the algorithm as a stream progresses. The memory usage (MB) on the Network Intrusion and Forest Cover data streams is reported. The memory usage on different window sizes of 1k, 2k and 5k is reported and these results are displayed in Fig. 3.7. For display purposes, each plot-point is an average over a set of windows. For windows of length 1k, 10 windows are averaged. For windows of length 2k, 5 windows are averaged, and for windows of length 5k, 2 windows are averaged. It can be seen that as the window size increases, the memory usage increases. Also, as the dimensionality increases (Forest Cover), the memory usage increases. Experiments were performed on a PC with an Intel processor at 2.6GHz and 8GB of RAM.

The reason ACSC performs faster than the comparative algorithms is because they each perform an exhaustive search for the nearest neighbour of each micro-cluster. If ACSC used a deterministic implementation whereby each point is compared with every other point, it would require O(N^2) time. But, each point in ACSC is evaluated against a sample taken from a cluster. The absolute worst case would require O(N^2) only if n data points in each window belonged to n different clusters. Results show that for the Network Intrusion stream, with 42 dimensions, the algorithm can process a window of 1,000 points in, on average, 0.04 seconds with an average memory requirement of 21.5 MB. The larger Forest-Cover stream can be processed in, on average, 0.08 seconds while requiring 37.6 MB of memory.

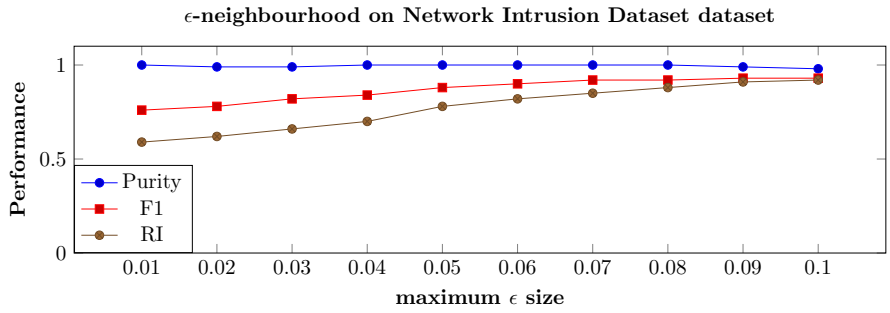


Figure 3.8: Sensitivity of ϵ-neighbourhood on Network Intrusion Stream.

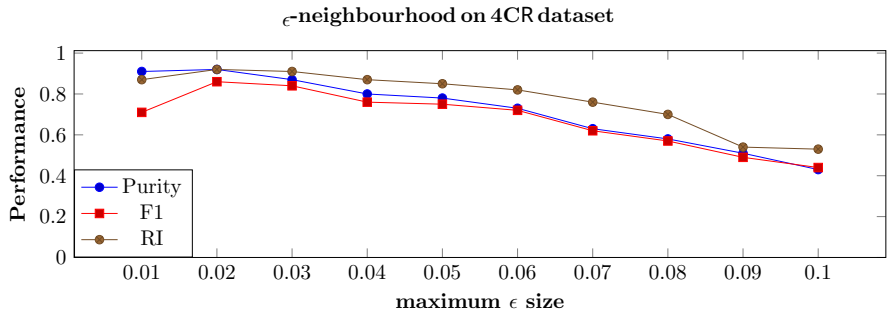


Figure 3.9: Sensitivity of ϵ-neighbourhood on $4CR$ Stream.

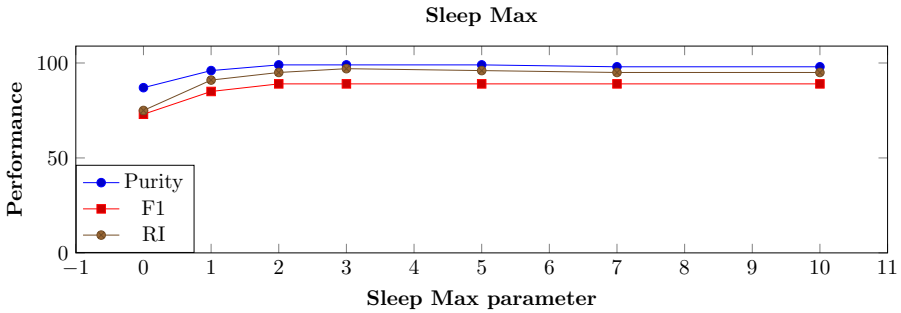


Figure 3.10: Sensitivity of $SleepMax$ on $4CR$ Stream.

3.6 Sensitivity Analysis

The sensitivity of the $nComp$ parameter was discussed in (Section 3.3). In this section the sensitivity of the ϵ parameter, the $sleepmax$ parameter, and the effect of different window sizes on the algorithm's performance is presented. The ϵ-neighbourhood is crucial in density clustering: if it is too small, no clusters will form; if it is too large, there will be rough and impure clusters. This value is sensitive and data-dependent. Figure 3.8 shows the sensitivity of the parameter on the Network intrusion stream while Figure 3.9 shows its sensitivity on the 4CR stream.

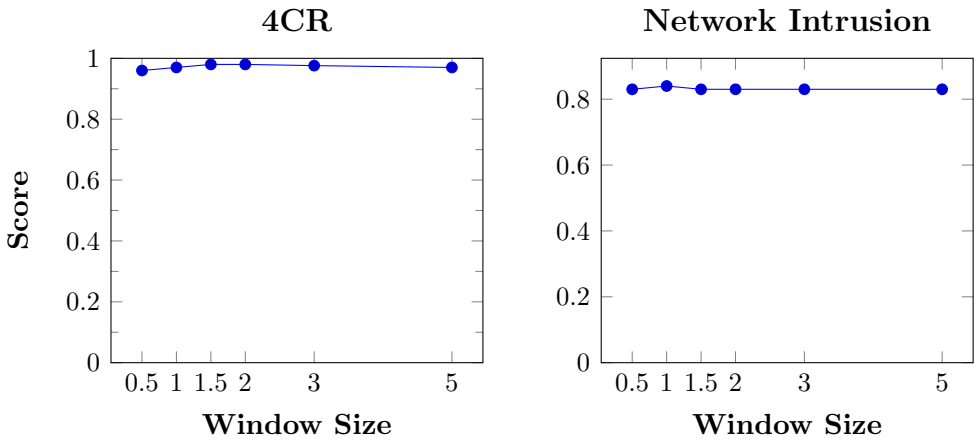


Figure 3.11: Sensitivity of window size (in thousands)

It can be seen that the parameter has a big impact on the clustering performance and it highly data-dependant; a larger value is better on the Network Stream while a smaller value is preferable on the 4CR stream. In contrast, the $SleepMax$ parameter is stable for all values greater than 0 (Figure 3.10).

To evaluate the sensitivity of window sizes, ACSC was tested across 6 different window sizes: 500, 1000, 1500, 2000, 3000 and 5000 on the Network Intrusion Stream. The Purity, F-Measure and Rand Index is calculated across each window in the stream and, for visualisation purposes, the 'score' is reported which is simply the average of all three metrics. The results are shown in Fig. 3.11. It can be seen that the window size has the minimal effect on the accuracy of the algorithm.

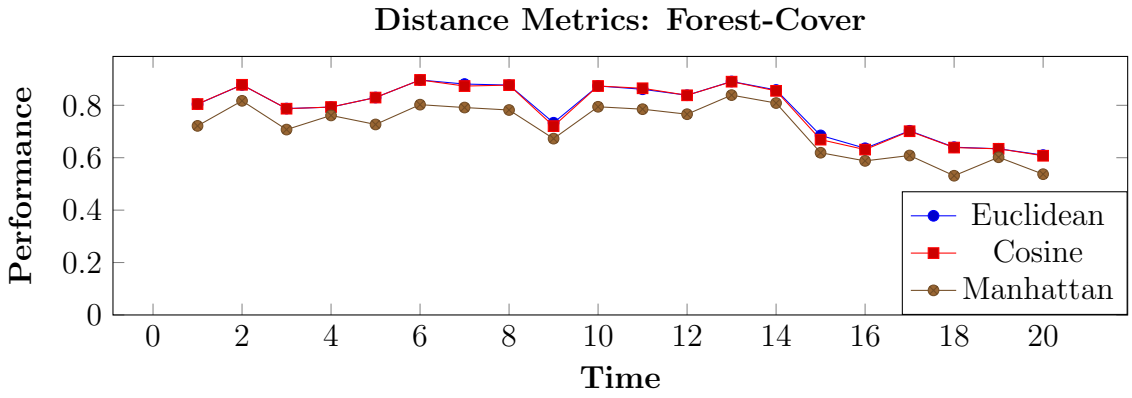


Figure 3.12: Different different metrics on Forest Cover Stream.

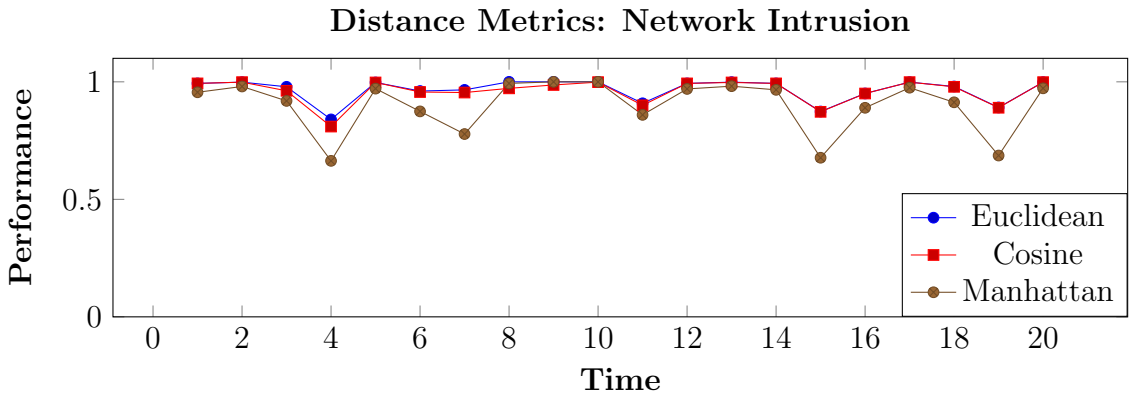


Figure 3.13: Different different metrics on Network Intrusion Stream.

3.7 Choice of Distance Metric

ACSC relies on geometric distance as the main clustering mechanic and there exists a number of different metrics for calculating this distance. In this section, three popular distance metrics are evaluated and compared; Euclidean, Manhattan, and Cosine distances. The distance between two points $X = \{x_1, x_2, \ldots, x_d\} \in \mathbb{R}^d$ and $Y = \{y_1, y_2, \ldots, y_d\} \in \mathbb{R}^d$, with d as the dimensionality can be defined as:

$$Euclidean(X,Y) = \sqrt{\sum_{i=1}^{n} \left(x_i - y_i^2\right)} \tag{3.18}$$

for the Euclidean distance.

$$Manhattan(X,Y) = \sum_{i=1}^{n} |x_i - y_i| \tag{3.19}$$

for the Manhattan distance, or by the Cosine distance :

$$Cosine(X,Y) = \frac{\sum_{i=1}^{n} X_i Y_i}{\sqrt{\sum_{i=1}^{n} X_i^2}\sqrt{\sum_{i=1}^{n} Y_i^2}} \tag{3.20}$$

To compare the effect of these distance measurements on the performance of the algorithm, two data streams are selected; the Network Intrusion Stream and the Forest Cover Stream. The first 20 windows of each stream (window length of 1,000 points) are evaluated using each metric and the average of the Purity, F-Measure, and Rand Index are taken at each time step and plotted as the stream progresses. Figure 3.13 displays the progress of the Network Intrusion Stream and Figure 3.12

displays the progress on the Forest Cover Stream. In both cases, the Euclidean distance returns the best result, although performance is only slightly better than the Cosine distance. It is interesting to note that the Cosine distance does not require square-root calculations so accepting a small drop in clustering performance could potentially further speed-up the algorithm.

3.8 Scalability and Robustness to Noise

3.8.1 Scalability

To test the scalability of ACSC, synthetic clusters of varying number and dimensions were generated. As in [31], the points in each synthetic data set are drawn

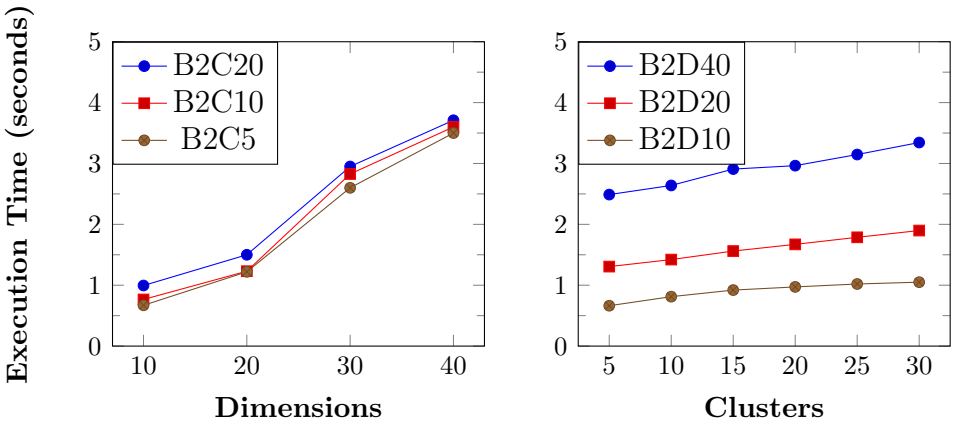


Figure 3.14: Scaling ACSC

Table 3.8: Wine with noise, $\epsilon = 0.07$

Noise	Purity	F-Measure	R. Index	#Nests
0%	0.94	0.9	0.88	4.1
3%	0.94	0.9	0.88	8.8
5%	0.93	.89	0.86	15.4
8%	0.91	0.88	0.87	20.1
10%	0.90	0.91	0.93	22.3
20%	0.91	0.65	0.70	38.5

from a series of Gaussian distributions (each representing a cluster). The mean and variance of each distribution are changed after every 5,000 points during the data generation process and the notion used here follows the notation used in [31] to describe the synthetic data sets: 'B' indicates the number of data points (in

Table 3.9: Network Intrusion with noise, $\epsilon = 0.09$

Noise	Purity	F-Measure	R. Index	#Nests
0%	0.99	0.95	0.94	1.6
3%	0.99	0.95	0.94	30.8
5%	0.99	0.93	0.94	52.0
8%	0.98	0.91	0.93	79.2
10%	0.98	0.91	0.93	90.3
20%	0.97	0.83	0.81	147.5

hundreds of thousands), 'C' and 'D' indicate the number of clusters present and the dimensionality of each point, respectively. For example, B1C20D10 indicates the data set contains 100,000 data points of 10 dimensions, belonging to 20 different clusters.

The performance of the algorithm is measured in the execution time using a window size of 10,000 and $\epsilon = 0.05$. The scalability of the algorithm, in terms of time, is evaluated against increasing number of clusters and also increasing number of dimensions.

First, the number of data points and clusters are fixed and the algorithm is tested on varying numbers of dimensions from 10 to 40. In Fig. 3.14, it can be seen that as the dimensionality increases, the execution time increases linearly. The plot follows a similar trend for each dataset regardless of how many clusters are present suggesting that the dimensionality is a more important factor than the number of clusters. This is confirmed when the number of data-points and dimensions are fixed but the number of clusters vary from 5 to 30. As the amount of clusters increases, the execution time increases only marginally.

3.8.2 Robustness to Noise

To evaluate how robust the algorithm is to noise, random noisy samples are introduced to two datasets; Wine, and Network Intrusion (first 100 windows of size 1,000). To introduce 5% noise, 5% of the final dataset is replaced with random samples. Tables 3.8, and 3.9 show the average performance over 30 runs of ACSC on each dataset with varying levels of noise. The results show that it is robust and the performance of the algorithm is not greatly affected by noise up to 10%. It is interesting to note that the number of clusters identified by the algorithm increases with the number of noisy samples. Each random point is assigned to its own cluster and the natural clusters remain relatively unaffected. With 20% of noise the performance of the algorithm drops though the algorithm maintains a high level of cluster

purity.

3.9 Summary

This chapter outlined Ant Colony Stream Clustering (ACSC). Results show that it scales linearly to larger window sizes and higher dimensionality, while being robust to noise. Clusters are formed in a single pass of the data using a stochastic sampling method. The sampling method replaces an exhaustive search and is shown to require considerably fewer calculations. The deterministic method (corresponding to $nComp$=1.0) yields the highest performance, at the cost of the longest run time. With a suitable choice of the parameter $nComp$, the proposed algorithm achieves a significant speed up at only little performance loss. The initial clusters discovered are further refined using a method inspired by the sorting behaviour of ants. This sorting method is based on the classic pick-and-drop ant clustering algorithm. The probabilistic functions for picking and dropping are biased towards the dissolution of smaller clusters and incorporating their contents into similar, larger clusters. This improves the precision and recall scores and creates clusters closer to the "true" structure of the data. This implementation addresses a short-coming of the original pick-and-drop model; speed. Rough clusters are identified quickly in a single pass and *then* sorted. Furthermore, in the traditional algorithm, data points are moved individually which can take a long time. By grouping similar points into micro-clusters, a number of points can be moved in a single operation, further speeding up the algorithm.

3.9.1 Limitations

Of the four required parameters for ACSC, three are shown to be insensitive to small changes and can be easily tuned. However, the ϵ parameter was shown to be very sensitive and greatly affects the performance of ACSC. This parameter defines the maximum radius of a micro-cluster and so determines the level of 'density' the user wishes to consider. It is data-dependent and requires manual fine-tuning. Furthermore, it is global and so restricts the algorithm to finding clusters of similar density, a common problem for density based clustering algorithms. This also restricts the *type* of clusters that can be discovered, for example overlapping or embedded clusters will be missed. A second limitation of ACSC is the algorithm's inability to label and track clusters as the stream progresses. The reason for this is the windowing method used. At each window clusters are discovered in a single pass and the input ordering of points greatly affects the order in which clusters are created. The clus-

ters (or their summary statistics) can manually be inspected to detect any change or pattern, however this cannot be automated with ACSC.

Chapter 4

Finding and Tracking Multi-Density Clusters

This chapter introduces the Multi-Density Stream Clustering algorithm (MDSC). This algorithm builds upon some of the main ideas introduced in ACSC (ants, nests, and a pheromone matrix) and address two key limitations of ACSC : 1) the inability to track clusters and 2) sensitive data-dependent parameters; specifically, the ϵ neighbourhood.

The ϵ-neighbourhood is the maximum radius permitted for each micro-cluster. As shown in Section 3.6 this is a sensitive parameter; if it is too large, multiple concepts will be clustered as one. If too small, no clusters will form at all. Furthermore, if this parameter is global, i.e., each cluster is constrained by the same value of ϵ (as in ACSC), then performance will degrade when the stream contains clusters of varying densities. As an example, a Gaussian source distribution that generates n points with a low variance will be more 'dense' than the same process with a large variance.

Recent proposals to capture multi-density clusters in a data-stream (for example MuDi [12]) rely on a number of sensitive user-defined parameters. The values of these parameters greatly affect the clustering performance and not much consideration is given to their practicality in a non-stationary environment. For example, how should these parameters be tuned? If we follow traditional methods, we could use a portion of the stream as a test set and use this to find the best set of parameters. However, in a dynamic environment, it is likely that the best values for these parameters will change over time.

This chapter is motivated by these challenges. In MDSC, clusters are discovered with an adaptive ϵ, local to each cluster. Each newly incoming data point is treated as a single micro-cluster which attempts to merge with existing, live clusters. If

the micro-cluster can not merge with an already discovered cluster, it attempts to merge with a micro-cluster in the outlier buffer. Otherwise, it joins the buffer as a new micro-cluster. This buffer is checked periodically for new clusters. A micro-cluster will age if no new data is added and it will eventually disappear if it is no longer relevant. This mechanism of ageing micro-clusters and an outlier buffer allows concept drift to be tracked and noise to be effectively treated. The interval at which the buffer is checked and the rate at which micro-clusters age are determined by a single user-defined parameter. A second user-defined parameter determines the age at which micro-clusters are considered no longer relevant and removed. These are the only user parameters and their values will depend on the velocity of the stream and the granularity the user wishes to examine it.

The ant metaphor and nest-building behaviour outlined in the previous chapter is extended here to discover new clusters in the buffer. In summary, the main contributions of this chapter are:

- The parameter ϵ in MDSC is adaptive and local to each cluster allowing for the discovery of clusters with varying densities.
- Discovered clusters are maintained online and labelled in order to track changes. Streams can be analysed in real time or at higher levels of granularity.

The work outlined in this chapter has been previously published in [57] and [58].

4.1 Multi-Density Stream Clustering

MDSC uses a time-dampened window model; processed data is subject to an ageing function and will disappear when it is no longer relevant. Also, the ϵ parameter which was global in ACSC is local to each cluster in MDSC. To allow for these changes the micro-cluster described in the previous chapter is extended. Previously a micro-cluster mc_i was defined by three components;

$$mc_i = [N, LS, SS] \tag{4.1}$$

where N is the number of points described by the micro-cluster and LS and SS represent the Linear Sum and Squared Sum respectively of each point in the micro-cluster. In MDSC, two additional components are used to describe a micro-cluster; *lastEdit* and a local ϵ. The *lastEdit* component is used to calculate the micro-cluster's age:

$$age = T - lastEdit \tag{4.2}$$

where T is the current time-step in the stream, and ϵ is discovered adaptively. So, in this chapter mc_i is defined by five components;:

$$mc_i = [N, LS, SS, lastEdit, \epsilon] \tag{4.3}$$

Micro-clusters still maintain the fundamental properties of merging and incrementing described in Section 3.1.1.

4.1.1 Finding New Clusters in the Buffer

During the initialisation step of the algorithm, λ points are collected in the buffer and the initial clusters are discovered. Any points not belonging to a cluster are retained in the buffer. Once the clusters are live, incoming data points are processed and those points which are not assigned to an existing cluster are passed to the outlier buffer. This buffer is periodically checked. Clusters are discovered in two steps; initially, micro-clusters form nests with similar micro-clusters and subsequently, similar nests are grouped to form the cluster.

Finding Nests

The step begins with a list of all micro-clusters currently stored in the buffer and a program variable ϵ-init. An appropriate value for ϵ is discovered adaptively using ϵ-init as an initial 'starting point', unlike in ACSC, where ϵ was a global, user parameter. This step proceeds in the same way as the first step in ACSC with two small differences. The minimum required similarity for an ant to join an established nest is ϵ-init which is typically much smaller than ϵ so a much larger number of initial nests will be created. Secondly, once all the ants have been assigned to their respective nests, the contents of each nest are merged into a single micro-cluster (Eq. (3.3)) with *no restriction on maximum radius* (i.e., $\epsilon = 1$). The pseudo-code for this step is presented in Algorithm 10. This is exactly the same as the first step of ACSC: each successive Merging ant joins or establishes a new nest and 'remembers' its similarity with each nest, thus creating and maintaining the pheromone matrix (PM). However, the final additional step (lines 17 - 19, Algorithm 10) is unique to MDSC. The difference between the two algorithms at this step is that ACSC is creating *macro*-clusters while MDSC is creating *micro*-clusters. Micro-clusters formed in this stage will vary in size, both in terms of radius and number of points contained. At the end of the step there are n nests, each containing a single micro-cluster and a PM describing the similarity between each pair of nests.

Algorithm 10 Create Nests

Input: List of micro-clusters in buffer, parameter ϵ-*init*
Output: Nests and Pheromone Matrix

```
1: for <each micro-cluster m> do
2:     if <nests> then
3:         for <each nest n> do
4:             Calculate similarity of m to n (Eq. (3.5))
5:             if <similarity ≥ ϵ-init > then
6:                 Add m to n
7:                 Update pheromone trail (Eq. (3.6))
8:             end if
9:         end for
10:    else if <No suitable nest> then
11:        Create a new nest
12:        Add m to the new nest
13:        Initialise pheromone trail
14:    end if
15: end for
16:
17: for <each created nest n> do
18:    Merge n into single micro-cluster
19: end for
20:
21: return Nests, Pheromone Matrix
```

Creating Clusters

The previous step summarised the buffer contents into a fewer number of heterogeneous micro-clusters, represented as a set of nests. In this step, clusters are discovered incrementally, starting with the most dense. This allows for the discovery of embedded and overlapping clusters.

A new cluster C is initialised with the densest nest in the set of nests as follows:

$$initialNest = \max_{k \in Nests} (k.N) \tag{4.4}$$

Take this nest as the initial nest, then find its closest neighbour in the pheromone matrix. These two nests are then merged into a single nest, the *seed* nest. Cluster C's ϵ-value is taken as the radius of this seed nest. Intuitively, clusters which are more sparse will have a greater distance between the initial nest and its closest neighbour (and consequently a larger ϵ). Conversely, more compact clusters will have a shorter distance and a smaller ϵ. Along with ϵ, cluster C requires an additional value, *threshold*, in order to group similar, density reachable nests which

Algorithm 11 Initialise Cluster

Input: Nests, Pheromone Matrix PM, Parameter α
Output: Cluster C

1: Find the densest nest $initNest$ in Nests (Eq. (4.4))
2: $initialDensity = initNest.N$
3: Find $initNest$'s closest neighbour $closestNest$
4: Merge $initNest$ and $closestNest$ into new micro-cluster $seed$
5: Initialise new cluster $C := seed$
6: $C.\epsilon := r_{seed}$
7: $C.threshold := initialDensity * \alpha$
8: Remove $initNest$ and $closestNest$ from Nests and PM
9: **return** C

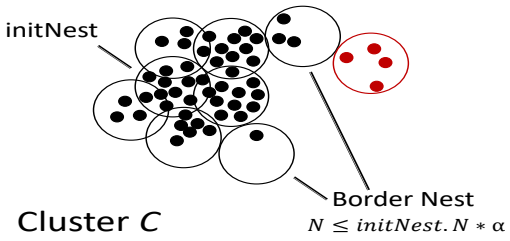


Figure 4.1: Illustrative example of cluster C in black. Although the red micro-cluster is density reachable to C, it is only via a border nest so does not become part of C.

remain in the buffer. This determines if a nest added to C is a *border* nest. A border nest is a nest which is density reachable to C but has a density (N) of less than α times the density of *initNest*. For example, if *initNest* contains 100 points and $\alpha = 0.1$, a border nest will contain 10 or fewer points. *Threshold* is local to C and relative to the density of C, controlled by a static program-variable α. Formally;

$$threshold = initNest.N * \alpha \tag{4.5}$$

This seeding process is outlined in Algorithm 11. Nests in the buffer which are density reachable to a border nest in C and *not reachable to any other nest in* C are not added to C. They remain in the buffer. This is illustrated in Fig. 4.1. This mechanism promotes the formation of homogeneous, pure clusters by preventing two (or more) similar concepts being clustered as one due to a small number of intermediary points. Furthermore, because nests are formed starting with the most dense (and therefore smallest ϵ), overlapping and embedded clusters can be discovered.

Once all nests which are density reachable to C are identified, the overall size of C is calculated - this is simply the number of data points described by C. If this size is greater than a minimum cluster size (proportionate to λ), C is added to the

Algorithm 12 Find Clusters

Input: Nests, Pheromone Matrix PM,
parameters *minClusterSize*, *clusterNum*
Output: Discovered Cluster(s)

```
1: newSeed := true
2: addedNest := true
3: while <Nests> do
4:     if <newSeed = true> then
5:         C := Initialise cluster (Algorithm 11)
6:         newSeed := false
7:     end if
8:     while <addedNest> do
9:         addedNest := false
10:        for <Each nest n> do
11:            if <n is density reachable to a non-border nest in C> then
12:                Add n to C
13:                Determine if n is a border nest (Eq. (4.5))
14:                Delete n from Nests
15:                Delete n from Pheromone Matrix
16:                addedNest := true
17:            end if
18:        end for
19:    end while
20:    if <C.size ≥ minClusterSize> then
21:        Merge micro-clusters in C (Eq. 3.3)
22:        C.label := clusterNum
23:        clusterNum++
24:        Add C to discovered clusters
25:        newSeed := true
26:    end if
27: end while
28: return Discovered Clusters
```

set of online clusters. If C contains fewer points than the minimum cluster size, the clustering operation is undone and the original nests remain in the buffer.

Before adding C to the set of online clusters, it is given a unique ID. This is a global parameter *clusterNum*, which is assigned to a cluster and then incremented, i.e., the first cluster is labelled as 1, the second as 2, and so on.

Outlier and noise points will unlikely ever get clustered and, because they are subject to an ageing process, they will eventually disappear in the buffer.

The pseudo-code for finding clusters from the initial nests is outlined in Algorithm 12.

4.1.2 Incoming Points

Online clusters are maintained as a set of connected micro-clusters. Each cluster has a unique id and a unique ϵ value. A newly arriving point p in d dimensions is first converted to a micro-cluster m as follows:

$$\begin{aligned} m.N &= 1 \\ m.LS_i &= p_i, \forall i \in \{1, 2, ..., d\} \\ m.SS_i &= {p_i}^2, \forall i \in \{1, 2, ..., d\} \\ m.lastEdit &= T \end{aligned} \tag{4.6}$$

The incoming micro-cluster attempts to join an existing cluster, checking each one beginning with the most compact, i.e., the cluster with the smallest ϵ value. The new micro-cluster checks if it is density reachable (Eq. (3.4)) to any micro-cluster in the selected cluster. If so, it attempts to merge (Eq. (3.3)). If the merging operation is a success, the merged micro-cluster's time-stamp is updated; otherwise, the newly-arrived micro-cluster is just added to the cluster (un-merged). If the newly arriving micro-cluster is not density reachable to any micro-cluster in any of the existing clusters, it is passed to the buffer. Here, it attempts to merge with a micro-cluster already present in the buffer; otherwise it joins the buffer as a new micro-cluster.

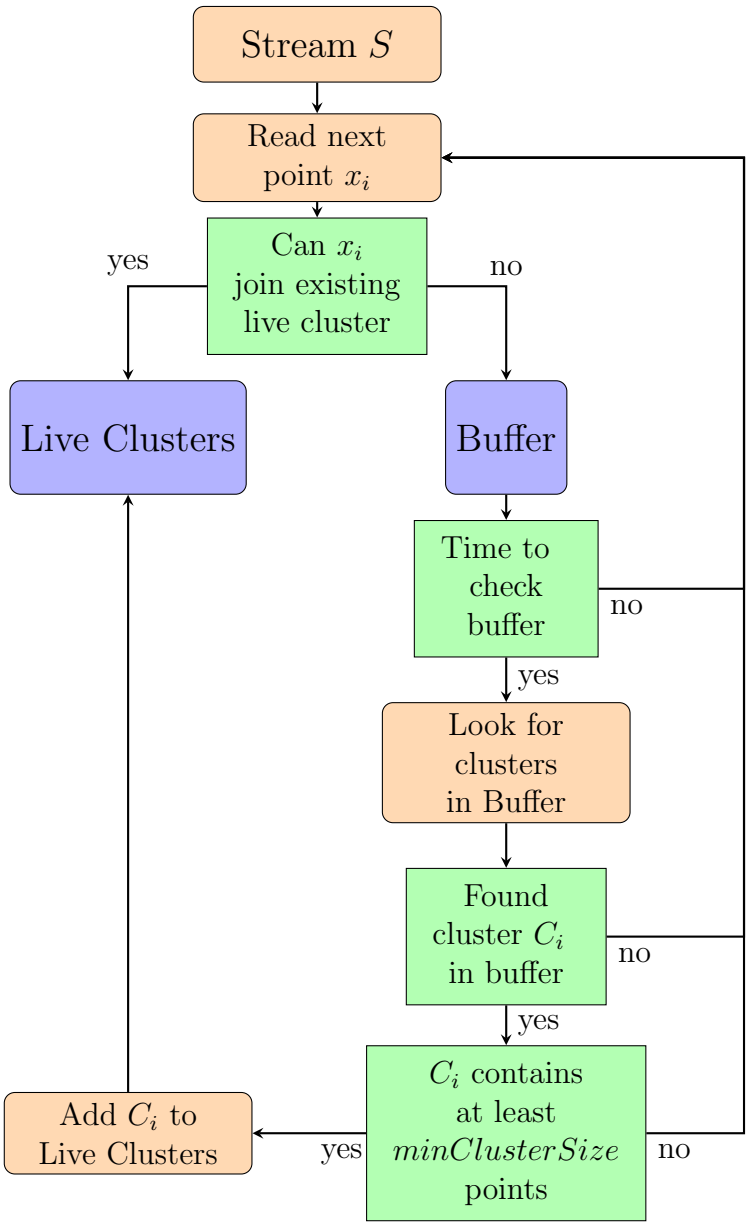
The entire process is presented as a flowchart in Figure 4.2.

4.2 Experimental Study

Four data-streams from different fields are selected in order to examine the performance of the algorithm without any parameter tuning. The performance of MDSC is compared with three peer density clustering algorithms. MuDi [12] is a recent algorithm proposed to handle multi-dimensional clusters. However, it requires a number of sensitive parameters and is not designed to track clusters online. CEDAS [87] is designed to track clusters online but is not capable of dealing with multi-density clusters. MDSC is also evaluated against ACSC. The algorithm's performance is then evaluated on synthetic data-streams exhibiting concept drift, concept evolution and clusters with varying densities.

The three metrics (Purity, F-Measure, and the Rand Index) described in Section 3.2.1 are used to evaluate the performance of each algorithm. MuDi and CEDAS are deterministic but ACSC and MDSC are stochastic. Each stochastic algorithm was run 50 times and the mean value is reported. To statistically evaluate the results, MDSC's scores are compared with the closest (better or worse) performance

Figure 4.2: MDSC Flowchart



from the peer algorithms. For the deterministic algorithm, the non-parametric One-Sample Wilcoxen Signed-Rank Test [196] is used and the null hypothesis that the distribution of MDSC's results are symmetric around the corresponding peer result is rejected with $p < 0.05$. To compare with ACSC the Wilcoxen Rank Sum test [196] is used and the null hypothesis that both results come from the same distribution is rejected with $p < 0.05$.

4.2.1 Datasets

The performance of MDSC is compared to the peer algorithms across seven benchmark datasets. Three datasets are taken from the Non-Stationary Environment Archive used in [178]. Two of these datasets are synthetic and are composed of non-stationary 2-D Gaussian clusters. One is $4CR$ (as previously described in Section 3.2.2) and the other is $2CSurr$ which is composed of two clusters with different densities. One cluster is stationary and the other is dynamic, exhibiting virtual concept drift (a change in $P(x)$). The third dataset taken from this archive is a real-world problem based on the use of keystroke dynamics to recognise users by the natural pattern of their typing rhythm, which is likely to change over time. It is based on 4 different users typing a 10-key password 400 times. The 10 variables measure the flight-time between each key, i.e., the time difference between a key being released and the next one being pressed, giving a total of 1,600 samples with 10 dimensions.

MDSC is also tested on the Network Intrusion and Forest Cover datasets previously described. Also included is a high-dimensional data-stream, COIL[1]. This is a dataset of 20 grey-scale images in 1024 dimensions (32 by 32 pixels) and exhibits concept evolution.

In order to evaluate the performance of MDSC on a stream exhibiting all of the challenging stream characteristics: concept evolution, real drift, virtual drift and multi-density clusters, a further synthetic data-stream with a dimensionality of 20 was generated. Clusters are Guassian and at periodic drift-intervals the centre and the variance of each one is shifted to simulate drift and varying densities. At each drift interval, a cluster is randomly added or removed (within bounds) to simulate concept evolution. In summary, MDSC is evaluated on 4 real and 3 synthetic data-streams exhibiting real concept drift (both sudden and gradual), virtual concept drift (gradual and sudden), concept evolution, and multi-densities. Streams with dimensionality ranging from 2 to 1,024 are evaluated and the details of each dataset is displayed in Table 4.1.

[1]http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php

Table 4.1: Description of datasets used in experiments. R = Real Data, VD = Virtual Drift, RD = Real Drift, MD = Multi-Density, CE = Concept Evolution

Dataset	Classes	Features	Examples	Characteristic
Real				
Network	5	42	250,000	R, VD
Forest	7	54	580,000	R, VD, CE
Key stroke	4	10	1,600	R, VD, MD
COIL	20	1,024	1,440	R,VD,CE,MD
Synthetic				
2CSurr	2	2	50,000	VD,MD
4CR	4	2	144,000	RD
20D	5-10	20	150,000	RD,VD,MD,CE

Table 4.2: Average performance on each stream measured using Purity (P), F-Measure (F), Rand Index (R)

	MuDi			*CEDAS*			*ACSC*			*MDSC*		
	P	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*
Network	0.97	0.87	0.81	0.99	**0.95**	**0.96**	**1.00**	**0.95**	0.94	0.99(s-)	0.93(s-)	0.94(s-)
Forest	0.73	0.47	0.52	0.86	0.48	0.59	0.88	0.59	0.64	**0.89**(s+)	**0.61**(s+)	**0.66**(s+)
KeySroke	0.61	0.46	0.70	0.87	0.61	0.67	0.88	0.56	0.68	**0.88**(=)	**0.65**(s+)	**0.77**(s+)
COIL	0.84	0.67	0.64	0.50	0.17	0.23	0.86	0.76	0.74	**0.92**(s+)	**0.81**(s+)	**0.81**(s+)
2CSurr	0.90	0.76	0.67	**0.97**	0.61	0.61	**0.97**	0.62	0.60	**0.97**(=)	**0.89**(s+)	**0.80**(s+)
4CR	0.94	0.94	0.91	0.98	0.95	0.96	**1.00**	0.95	0.97	**1.00**(=)	**0.98**(s+)	**0.98**(s+)
20D	0.92	0.87	0.94	0.98	0.79	0.93	0.96	0.77	0.93	**0.99**(s+)	**0.94**(s+)	**0.97**(s+)
Average	0.84	0.72	0.74	0.87	0.65	0.7	0.93	0.74	0.78	**0.94**	**0.83**	**0.84**

4.2.2 Clustering Quality Evaluation

The comparative results are presented in Table 4.2. On the four real data-streams MDSC, on average, performs better than each peer algorithm despite requiring no parameter tuning for each specific stream.

Parameters for the peer algorithms are tuned (using the first n points in the stream as a training set) on each specific stream. On each stream a value of 4 is used for β, so micro-clusters which have not been updated in 4 λ intervals are removed ($\lambda = 1000$). Overall, purity levels are comparative with ACSC but each of three peer-algorithms are outperformed on the F1 and Rand Index metrics. As these are real datasets, it is difficult to tell whether MDSC outperforms the peer algorithms because of the clustering mechanism itself or because the streams contain different densities. From the adaptive ϵ values, we can infer that *Network* and *Forest* contain a single level of density (*Forest* ≈ 0.2 and *Network* ≈ 0.11).

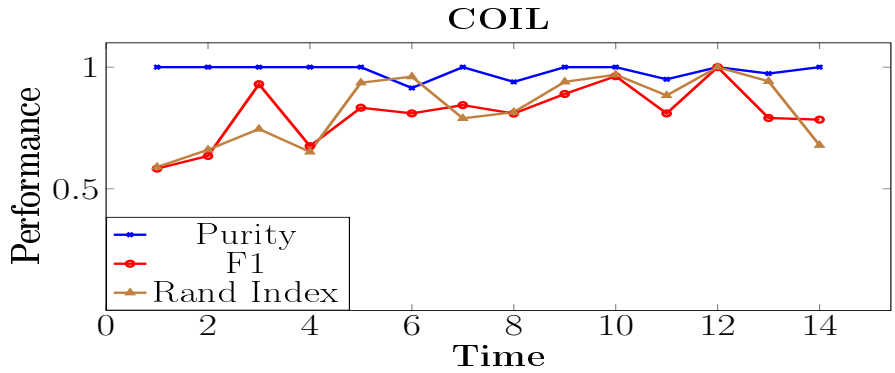


Figure 4.3: Progression of the COIL stream.

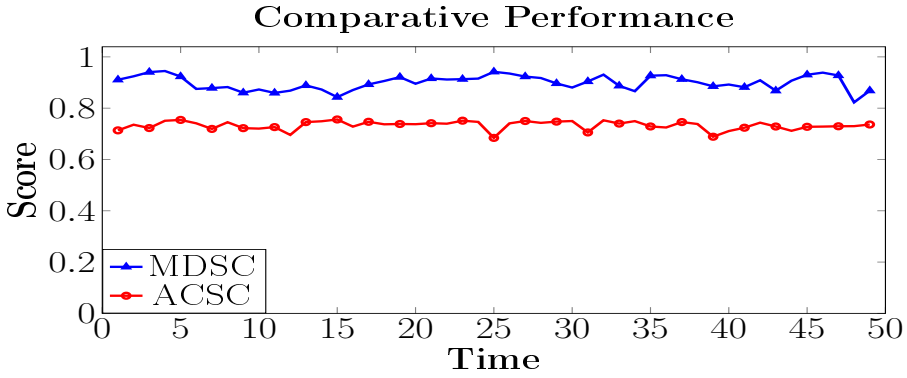


Figure 4.4: Comparative Performance on the 2CSurr stream.

However, on *COIL* and *Keystroke*, MDSC finds clusters with varying densities (0.005 to 0.26 on *COIL*, 0.002 to 0.01 on *keyStroke*). The progression of the COIL stream using the three metrics is presented in Fig. 4.3.

To evaluate on streams that certainly contain different densities, two synthetic streams: $2CSurr$ and $20D$ are used. The comparative results are displayed in Table 4.2. MDSC outperforms the others on each of the three metrics. $2CSurr$ consists of two clusters (one stationary and one dynamic) with different densities. The dynamic cluster is much more compact and this is reflected in its ϵ-value of 0.012, much smaller than the other cluster with a density of 0.029. The comparative performance with ACSC on this stream is presented in Fig. 4.4. For illustrative purposes, 'score' is reported; this is simply the average of Purity, F1 and Rand Index.

4.3 Parameter Tuning in Non-Stationary Streams

In the previous comparison with ACSC, an ϵ value of 0.02 was used. The second synthetic stream $20D$ is used to illustrate the problems with tuning a sensitive

Table 4.3: Tuning the ϵ Parameter

ϵ	1,000	5,000	Full Stream
0.1	0.62	0.72	0.73
0.09	0.62	0.72	0.74
0.08	0.60	0.75	**0.77**
0.07	0.62	**0.76**	0.74
0.06	0.63	0.74	0.73
0.05	0.63	0.74	0.74
0.04	0.65	0.68	0.72
0.03	**0.67**	0.62	0.70
0.02	0.66	0.55	0.69
0.01	0.47	0.50	0.64

parameter such as this in a dynamic stream. $20D$ contains between 2 and 10 non-stationary clusters in 20 dimensions. The stream exhibits concept evolution, concept drift (both real and virtual) and clusters with varying densities. A "training" set (the first n samples) was taken and the remaining stream was used as the "test". The results of ACSC with different ϵ values are displayed in Table 4.3, the result in each case is the average of purity, F-Score, and Rand-Index. If we take the first 1,000 points as a test, the best value for ϵ is 0.03; if we take a test set of 5,000 points, the best value is much larger at 0.07. Neither value gives the best performance over the entire stream; in this case the value that gives the best performance would be 0.08. This has implications in both the practicality of parameter tuning and also, which final result should be reported as a measure of the algorithm's performance.

4.4 Tracking Clusters

To illustrate how discovered, online clusters can be tracked and their drift observed, two 2D data-streams are selected for illustrative purposes: $2CSurr$ and $4CR$. $2CSurr$, as previously described, contains two clusters with varying densities and displays virtual concept drift. $4CR$ consists of 4 rotating clusters. The clusters rotate into positions previously occupied by a different cluster, showing real drift.

To illustrate this drift, the centre of discovered clusters are recorded at each time-step and then displayed in a scatter plot. In the $2CSurr$ stream, both clusters were tracked and their trails are displayed in Fig. 4.5. In 4CR, the clusters share a similar level of density, with $\epsilon = \{0.02, 0.021, 0.022, 0.025\}$. The positions of each are displayed over the first 50,000 points in the stream. Clusters rotate into positions

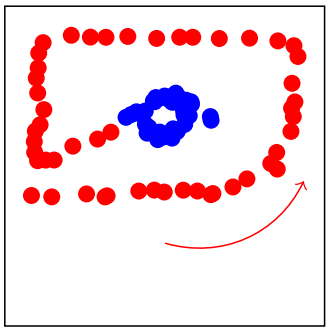


Figure 4.5: Drift in 2CSurr stream. Blue cluster is stationary and red cluster drifting in the direction of the arrow. Center of clusters are recorded every time-step and the drift is captured and tracked.

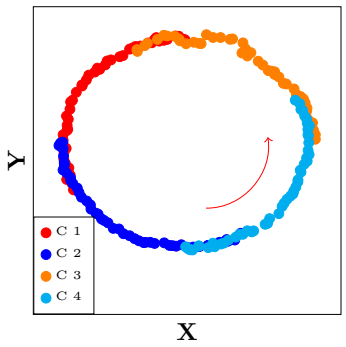


Figure 4.6: Drift in 4CR. The first 50,000 samples are presented. Underlying change represents a shift in $P(y|X)$, the conditional probability of cluster y given position X. This underlying shift is tracked and can be seen in the overlap between each cluster's trail.

previously occupied by a different cluster so the underlying change represents a shift in $P(y|x)$, the conditional probability of cluster y given position x. This underlying shift is tracked and can be seen in overlap between each cluster's trail in Fig. 4.6.

4.5 Complexity Analysis

This section examines the algorithm's complexity and empirical results are reported on two real data-streams; *Network Intrusion* and *Forest Cover*. In the following section N refers to the number of clustered micro-clusters (in live clusters) and M refers to the number of micro-clusters in the outlier buffer. Typically, M is much smaller than N.

The time complexity of the algorithm depends on the value of λ as this determines how frequently buffer is examined for new clusters. The complexity of joining a live cluster is $O(N)$; a newly arriving point tests if it is density reachable with every micro-cluster in the discovered clusters. Periodically, when the buffer is examined, the processing time will increase. This increase is a function of change and noise. If there is no change or noise, then the buffer is empty and no further processing is required. If, however, the buffer is not empty, then the processing time increases to, in the worst case, $O(M^2)$ (The nest-building stage of the buffer check requires $O(M^2)$ and the clustering phase requires $O(\log M)$). The time complexity of MDSC is $O(N)$ and at every λ intervals it requires an additional $O(M^2)$. In total, the algorithm requires $O(N + \frac{M^2}{\lambda})$.

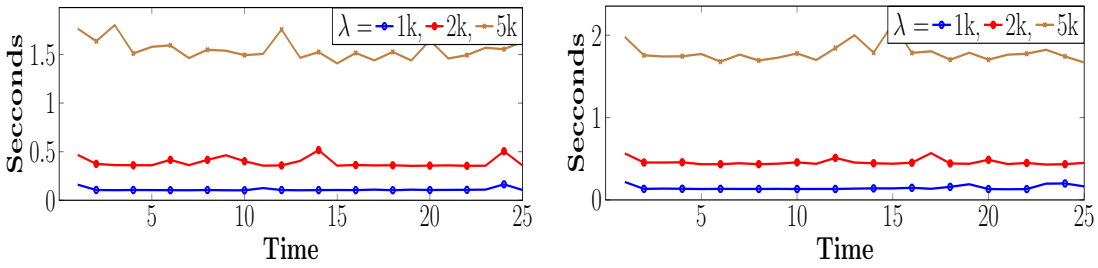


Figure 4.7: Time requirements on Network Intrusion (left) and Forest-Cover (right) using different λ values.

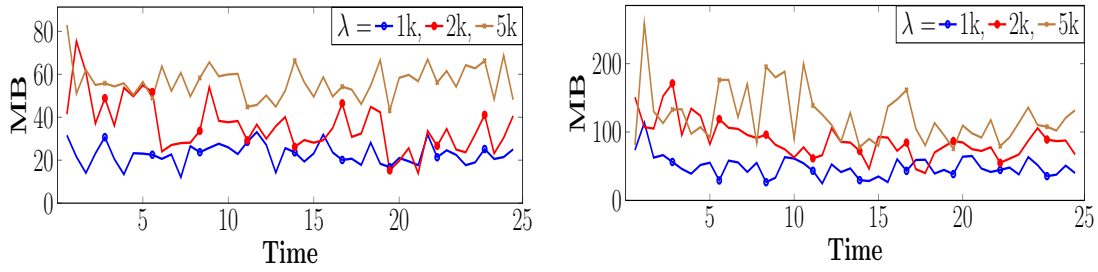


Figure 4.8: Memory requirements on Network Intrusion (left) and Forest-Cover (right) using different λ values.

Space is measured in terms of the number of micro-clusters that have been clustered plus the number in the outlier buffer: $O(N + M)$.

The algorithm's performance is empirically measured using different λ values. Large values mean micro-clusters will age more slowly, the buffer checked less regularly, and therefore a greater number of micro-clusters. The time requirement is reported in seconds (Fig. 4.7) and the memory requirements in MB (Fig. 4.8). As in the previous chapter, a commercial profiler [90] is used to measure the memory usage as the stream progresses. For clarity of display, the first 250,000 points in each stream are plotted. For $\lambda = 1,000$ the average of 10 windows is plotted (25 points), for a value of $2,000$ the average of 5 windows is plotted (25 points) and for $\lambda = 5,000$ 2 windows are averaged.

On the Network Intrusion Stream, the algorithm can process 1,000 points in, on average, 0.11 seconds, requiring, on average, 22MB. This rises with large values of λ. Similar results are seen on the Forest Cover stream. It requires 0.14 seconds to process 1,000 points requiring 41MB, on average.

4.6 Sensitivity Analysis

To examine the sensitivity of the algorithm to its parameters, it is evaluated across four streams: *Forest Cover*, *Network Intrusion*, *2CSurr* and *4CR*. These streams

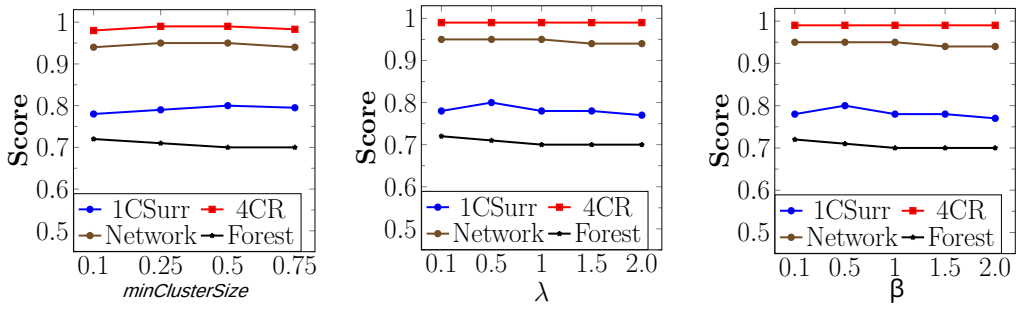


Figure 4.9: Sensitivity of *minClusterSize*, λ and β.

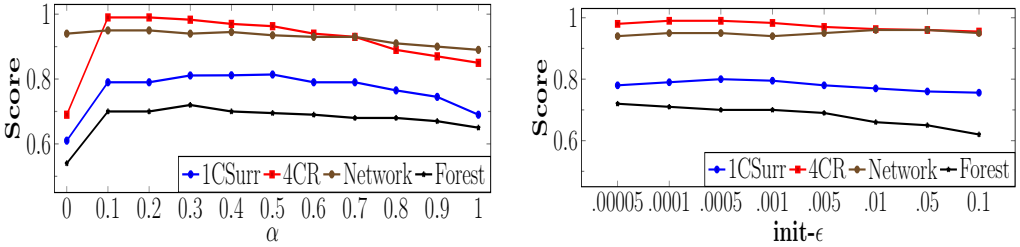


Figure 4.10: Sensitivity of α and $init-\epsilon$ program variables.

are selected in order to cover real data streams (*Network, Forest*), real concept drift (*4CR*) and multi-density clusters (*2CSurr*). MDSC requires two user-defined parameters λ and β: λ determines the rate at which micro-clusters age and the frequency at the which the buffer is examined while β determines the age at which micro-clusters become irrelevant. These parameters determine the granularity at which the stream is analysed and should be judged according to the velocity of the stream, e.g., a stream with one point a second versus a stream with 1,000 points a second.

The sensitivity of these parameters to cluster quality is presented in Fig. 4.9. It can be seen that over a range of values the *quantative* performance is unaffected. The *qualitative* values of the clusters will change though. For example, at a higher λ, long-term patterns will be discovered but smaller changes will be missed. Conversely, a smaller λ is more sensitive to change but will miss broader patterns. Along with these two user-defined parameters, there are three program values. *minClusterSize* determines the minimum size a cluster discovered in the buffer must be in order to be added to the live clusters. This is proportionate to the size of λ. For a smaller λ, the buffer is checked more frequently so smaller clusters should be allowed. When the buffer is checked infrequently, there will be potentially more instances in the buffer, so clusters are required to be larger. Fig. 4.9 (left) shows this parameter to be robust to values above 0 across each data stream tested. For all experiments, a value of 1% of λ is used with a minimum value of 2.

The parameter ϵ-init determines the initial value for ϵ when forming nests in the buffer and is used as the initial 'starting point' for the adaptive ϵ for each cluster. From Fig. 4.10 (right), it can be seen that, for values less than 0.01, the performance is stable across each stream. For all experiments in this chapter, a value of 0.001 is used. The final program parameter is the α threshold value, which defines a border nest in a cluster. This parameter is relative to the number of points clustered in the seed nest of a cluster. A larger seed will have a higher value for α. From Fig. 4.10 (left), we can see that the performance is stable across all streams with a value greater than 0 and less than 0.7. For all experiments discussed, a value of 0.1 is used.

4.7 Scalability and Robustness to Noise

4.7.1 Scalability

To evaluate the algorithm's scalability, synthetic data sets with varying dimensionality and number of clusters are generated. As in the previous chapter (Section 3.8), the instances in each synthetic stream are drawn from a series of Gaussian distributions (each representing a cluster). The mean and variance of each distribution are shifted every 5,000 points during the generation process. The same notation is used to describe the streams: 'B' indicates the number of data points (in hundreds of thousands), 'C' and 'D' indicate the number of clusters present and the dimensionality of each point, respectively.

The performance of the algorithm is measured in the execution time using a λ value of 10,000 and $\beta = 5$. The scalability of the algorithm, in terms of time, is evaluated against an increasing number of clusters and also increasing number of dimensions. In Fig. 4.11 (left), it can be seen that as the dimensionality increases, the execution time increases linearly, irrespective of the number of clusters, suggesting that the dimensionality is a more important factor than the number of clusters. This is confirmed when the dimensionality is fixed and the number of natural clusters is increased from 5 to 30, see Fig. 4.11 (centre). The stream takes roughly the same processing time as the number of clusters increases, with higher dimensional streams taking longer than lower dimensional ones. As the amount of clusters increases, the execution time increases only marginally.

As in the previous chapter, the space requirements of MDSC is evaluated in terms of micro-clusters present at different λ intervals. Three previously described data streams are used to evaluate this; 2CSurr, 4CR and the Network Intrusion stream. Fig. 4.11 (right) shows that as λ increases from 1,000 to 5,000, the number

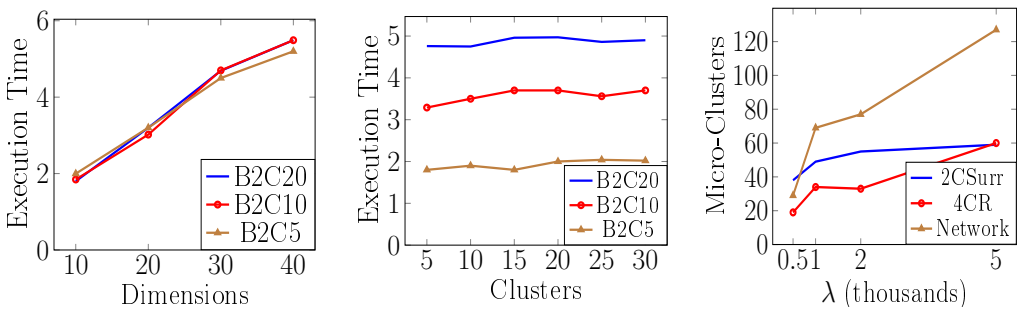


Figure 4.11: Scaling to the number of clusters, dimensions and memory requirements.

Table 4.4: Noise Sensitivity on the Network Stream

Noise	Purity	F-Measure	R. Index	Average#Nests
0%	0.99	0.91	0.88	2.14
3%	0.99	0.91	0.88	2.16
5%	0.99	0.91	0.87	2.2
8%	0.99	0.91	0.88	2.11
10%	0.99	0.91	0.88	2.05

Table 4.5: Noise Sensitivity on 4CR

Noise	Purity	F-Measure	R. Index	Average#Nests
0%	0.99	0.94	0.96	4.12
3%	0.99	0.94	0.96	4.08
5%	0.99	0.94	0.95	4.09
8%	0.99	0.94	0.95	4.07
10%	0.99	0.93	0.95	4.04

of micro-clusters generated increases, at most, linearly on the Network Intrusion Dataset. On the *4CR* and *2CSurr* streams, the change is comparatively small as λ increases.

4.7.2 Noise

To evaluate how robust the algorithm is to noise, random samples are introduced to two datasets; Network Intrusion (first 100,000 samples) and 4CR. To introduce 1% noise, 1% of the final dataset is replaced with random samples and evaluated with a λ value of 1,000 and a β of 5. Streams with varying levels of noise across the three external metrics are evaluated and the average number of live clusters in the stream is reported. The results on the two datasets are displayed in Tables 4.4 and 4.5, respectively. It can be seen that as the amount of noise increases the performance

of the algorithm remains stable, and the number of live clusters remains relativity constant. This is because noise points are based to the buffer where the remain until they age and are removed, never joining the live clusters.

4.8 Case Study: Leicester Air Quality

The previous data-steams are all labelled and MDSC's performance could be measured objectively according to the known ground-truth. In this section, its performance is *quantitatively* evaluated on a real-world data stream in order to assess the utility of the discovered clusters. A real-life stream is taken from UK-AIR database maintained by the Department of Environment, Food and Rural Affairs (DEFRA)[2]. The data-stream is taken from a single monitoring site in Leicester City and provides seven hourly air-quality measurements regarding Ozone (O_3), Nitric Oxide (NO), Nitrogen Dioxide (NO_2), Nitrogen Oxides (NO_x), Particulate Matter 2.5 ($PM_{2.5}$), Non-volatile PM2.5 (PM_{nv}), and volatile PM2.5 (PM_v). These seven pollutants are measured in micrograms (one-millionth of a gram) per cubic meter air or $\mu g/m^3$. Data was collected from January 1st 2014 to April 1st 2017 and each point is read in time order to simulate the original stream. This stream is used to study the performance of MDSC only and not as a case-study in air-quality in Leicester since such a study would require data from a wider range of sites and would need to factor external influences such as weather and wind, etc.

4.8.1 Internal Evaluation Metric

When evaluating MDSC on this stream the external metrics (Purity, F-Measure and Rand Index) cannot be used as there are no associated labels. To overcome this issue, an internal evaluation metric: the Silhouette Coefficient (SC) [164] is used.

The SC is a measure of how similar a data instance is to its own cluster (cluster cohesion) compared to instances in other clusters (cluster separation). For each instance i A denotes the cluster to which i belongs. The average similarity ($a(i)$) of i to all other instances in A:

$$a(i) = \frac{1}{\|A\| - 1} \sum_{j \in A, j \neq i} dist(i, j) \tag{4.7}$$

[2]https://uk-air.defra.gov.uk/data/

Next, for any cluster C which is not A, the average similarity from i to C is:

$$d(i, C) = \frac{1}{\|C\|} \sum_{j \in C} dist(i, j) \tag{4.8}$$

After finding $d(i, C)$ for all clusters ($C \neq A$), take the *minimum* distance $b(i)$, formally

$$b(i) = min(d(i, C)) \tag{4.9}$$

The cluster B with this minimum value is referred to as the Silhouette of A. The SC value $s(i)$ is calculated as follows:

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))} \tag{4.10}$$

The SC value for the whole cluster A is the mean of all instances i in A and the SC for the clustering solution R (where $R = \{K_1, K_2, ..., K_n\}$) is the mean of all Silhouette values in R (i.e., $\frac{1}{n} \sum_{i=1}^{n} s(K_i)$). The SC value lies between -1 and 1, where 1 represents a good clustering solution.

Like all real-world sensors, some data is missing. If an entire reading (all seven variables) are missing, the point is read (in order for continuity) but no attempt is made to cluster the point. If a single variable is missing it is estimated by taking the average of the hour before and an hour after.

4.8.2 Analysis at a Weekly Granularity

To evaluate the stream on a weekly level λ is set to 168 (24 hours by 7 days) and β to 4. So, micro-clusters age every week and a micro-cluster which has not been updated for 4 weeks (roughly 1 month) is considered no longer relevant and removed. At each time-step, the mean, min and max values of each live cluster (i.e., the mean, min and max of the cluster's constituent micro-clusters) are recorded and stored off-line for evaluation. The stream is examined at a weekly level but λ and β could be set to different values for a monthly granularity (say, $\lambda = 720$, i.e., 24 hours by 30 days), daily ($\lambda = 24$), or real-time ($\lambda = 1$).

In week one, three clusters were discovered and their mean values are displayed in Table 4.6. Cluster 1 has high levels of Ozone (O_3) but comparatively low levels of Nitrates (NO_x) and Particulate Matter ($PM_{2.5}$). Cluster 3 has a quarter of the levels of O_3 but higher levels of NO_x and $PM_{2.5}$. Cluster 2's levels lie between clusters 1 and 3. This suggest an inverse relationship between O_3 and NO_x and $PM_{2.5}$. Over the course of the stream (171 weeks), between 2 and 6 clusters are active each

Table 4.6: Initial Clusters Discovered in Week 1

Cluster	O_3	NO	NO_2	NO_x	PM2.5	PMv	PMnv
1	63.06	2.92	15.03	19.50	8.13	5.39	2.71
2	30.96	12.59	44.44	63.75	7.677	5.12	2.42
3	14.29	22.57	63.37	97.74	12.63	10.58	1.94

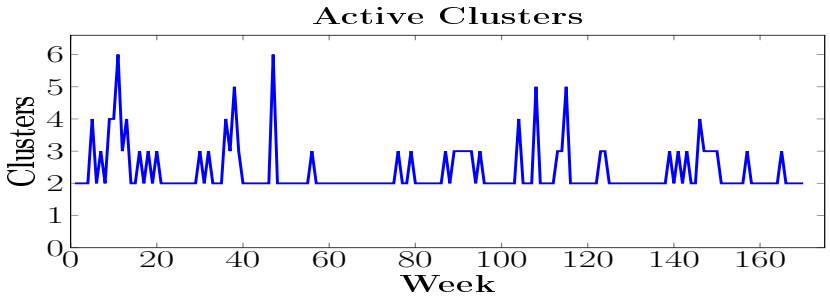


Figure 4.12: Active Clusters Each Week.

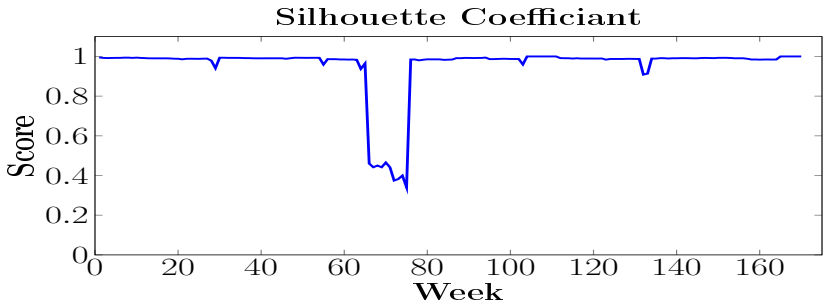


Figure 4.13: Silhouette Coefficient on Air Quality Stream.

week. 'Active' is used to mean that at least one new instance during the week was added to a live cluster. Clusters can still be live but inactive (no added points). The number of active clusters is displayed in Fig. 4.12 and the corresponding Silhouette Coefficient (SC) values are presented in Fig. 4.13.

The high SC scores suggest the live clusters are well separated and cohesive except for a 'wobble' at approximately week 60. Of the active clusters, clusters 1 and 3 are active throughout and represent the main underlying pattern. Their levels of O_3, NO_x and $PM_{2.5}$ are presented in Fig. 4.14. NO_x includes NO and NO_2, and $PM_{2.5}$ is the combined total of volatile and non-volatile $PM_{2.5}$. So the seven measured variables are reduced to three just for clarity of display.

Looking at the progression of cluster 1, we can observe seasonal changes in the measured atmospheric gases. Levels of O_3 are higher in summer than in winter, the inverse of NO_x. These seasonal changes are tracked. Cluster 3 shows an exaggerated version of the same trend, much higher levels of NO_x and $PM_{2.5}$ but lower levels of

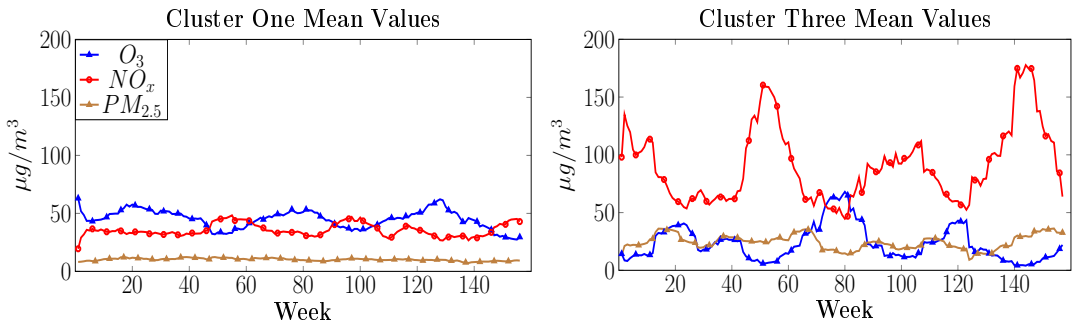


Figure 4.14: Mean Values of Persistent Two Clusters in Air Quality Stream.

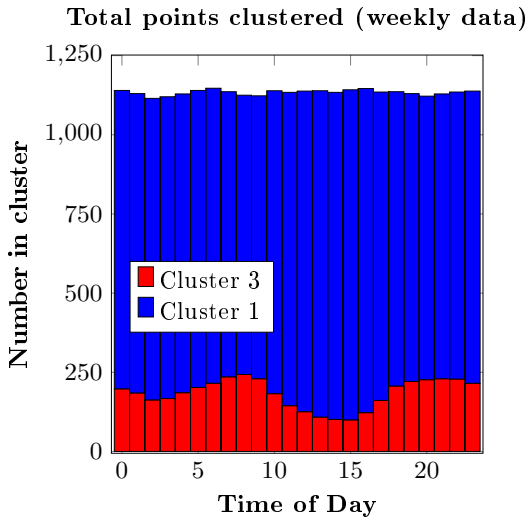


Figure 4.15: Relative Sizes of Cluster 1 and 3 and the hours they are most active.

O_3.

It has been observed [89] that in an abundance of NO_x, O_3 is 'scavenged' as it reacts with NO_x so this could explain the symmetry of the two levels in Cluster 3. The trends also suggest a correlation between NO_x and $PM_{2.5}$. Both are comparatively low in cluster 1 and higher in cluster 3.

Clusters 1 and 3 capture the main underlying pattern of the stream. Cluster 1 represents a pattern of low levels of air pollution and is the largest cluster throughout. The relative sizes of each are presented in Fig. 4.15 along with the time-of-day they represent. The x-axis displays the 24 hours in a day and the y-axis represents the total number of potential hourly instances in the stream, in this case 1,197 (171 weeks $\times$ 7 days, 1,197 instances of 10am, for example). Cluster 1 can be seen to be much larger. For Cluster 3, its higher levels of NO_x and $PM_{2.5}$ occur more frequently at around 8am and 6pm, typically rush-hour in a city. From this, we can infer that cluster 1 represents the 'usual' levels of air quality, cluster 3 represents the rush-hour pattern and the remaining number of instances are distributed in clus-

ters that represent anomalies or change. For example, cluster 5 was discovered in week 7 and represented a pattern of high particulate matter (≈ 26). This cluster was present until week 126 when it disappeared and was not replaced suggesting a change in the stream.

4.9 Summary

MDSC extends some of the swarm intelligence inspired aspects of density clustering introduced in ACSC, e.g., pheromone trails and the idea of micro-clusters forming 'nests'. MDSC improves ACSC in two ways: clusters are online and the ϵ parameter is adaptive and local to each cluster. The adaptive ϵ has two benefits: 1) it removes the need to tune a sensitive, data-dependent parameter, and 2) it allows the discovery of multi-density clusters. This can be seen in the comparative performance; on the streams which contain multi-densities (*COIL*, *Key Stroke*, *2CSurr* and *20D*), MDSC finds a better clustering solution. When the stream contains a single density (*Network*, *Forest*, *4CR*), the performance is comparable to ACSC. The reason for this is the adaptive ϵ and the way it is identified. Clusters are discovered in order of density, i.e., more compact clusters are identified first. This allows the discovery of clusters (with a smaller ϵ) embedded in larger sparser clusters (with a higher ϵ). Discovered clusters are uniquely labelled (cluster 1, cluster 2, ...) and can therefore be tracked over time. This was shown in the 2D synthetic data sets; the underlying drift in the stream was discovered and tracked. This was further illustrated in the case study on the Leicester air-quality. Discovered clusters were tracked despite seasonal changes in the monitored atmospheric gases. Underlying patterns were discovered along with their deviations, and cyclic patterns were revealed. These live, labelled clusters allow us to infer a broader picture of what is happening in the stream and we can do this at different granularities. In the case study, the stream was examined at a weekly granularity. The granularity is controlled by two parameters: λ, which determines the rate at which data ages, and β, which determines how long data is relevant. Changing these two parameters dictate the granularity at which the stream is analysed. Analysing a stream at different granularities in parallel could catch brief anomalies while revealing the broader behaviour of the stream.

Chapter 5

Dynamic Feature-Selection and High-Dimensional Streams

Change in a data stream can occur at both concept and feature levels. The previous chapters focused on clustering streams in the presence of change at the concept level but not at the feature level. Furthermore, the methods proposed (in the previous chapters and also the wider literature) often rely on distance as a similarity metric and this is problematic for high-dimensional data where the curse of dimensionality renders distance measurements and any concept of 'density' difficult. This chapter proposes a solution to these two problems by combining them and framing the problem as a feature selection problem. Specifically, a *dynamic* feature selection problem.

Feature selection (FS) aims to identify a subset of the most relevant features $\hat{f}$ from the set of all features F. Traditionally, $\hat{f}$ would be used to cluster data and all redundant features ($\{f_i : f_i \in F \text{ and } f_i \notin \hat{f}\}$) are ignored for future points. This might not be a sensible approach to non-stationary data as $\hat{f}$ is likely to change over time. A significant change could require previous clusters to be abandoned and new clusters discovered on the latest data. This would be especially true for clustering algorithms that rely on some form of distance as a similarity metric; it might not be possible to cluster two points composed of different feature subsets, e.g., if the number of 'important' features changes ($|\hat{f}_t| \neq |\hat{f}_{t+1}|$) or a previously important feature is no longer considered important ($f_i \in \hat{f}_t$ but $f_i \notin \hat{f}_{t+1}$)).

Motivated by these challenges this chapter introduces a dynamic feature mask for clustering high dimensional data streams. The proposed method is algorithm-independent and can be used with any density-based clustering algorithm which typically has no mechanism for dealing with feature drift and struggles with high-dimensional data. The method is designed to 'sit on top' of any existing density-

based stream clustering algorithm.

A stream is split into windows and unsupervised FS is performed after each window. Redundant features are masked and clustering is performed along unmasked, relevant features. If a feature's perceived importance changes, the mask is updated accordingly - previously unimportant features can be unmasked and features which loose relevance become masked. As new features appear in the stream, the size of the mask is changed. Clustered points contain all features (not just a subset of relevant features) but the clustering process only considers the subset of relevant features.

In summary, a novel Dynamic Feature Mask method for clustering high dimensional data-streams is outlined and the main contributions of this chapter are:

- Feature Drift and Feature Evolution can be detected and tracked in a fully unsupervised way and the importance of features can be monitored over time.
- The method is algorithm-independent and can be used with any of the existing density-based stream clustering algorithms which typically do not have a feature drift mechanism and are unable to deal with high dimensional data.
- Applied to an existing stream clustering algorithm, the proposed method can reduce the time requirements and increase accuracy.

The work outlined in this chapter has been submitted for publication [59].

5.1 Dynamic Feature Mask

In the proposed method, a feature mask is maintained and clustering is performed according to this mask. A stream of instances arrive online. When a point arrives, it is passed to the clustering algorithm and, also, a copy of the point is stored in an offline buffer. When the buffer reaches a pre-defined size $bufferSize$, feature selection is performed on the buffer and the feature mask is updated. This mask is used for the clustering process until the next $bufferSize$ points arrive in the stream. This chunk is referred to the β-window in order to differentiate it from the sliding window used by the underlying clustering algorithm to process the stream. These two windows do not necessarily need to be the same size.

5.1.1 Preliminaries

The proposed method requires an unsupervised feature selector. Three existing *static* methods are evaluated for maintaining the *dynamic* feature mask. The Lapla-

cian Score (LS), Maximum Variance (Var) and Multi Cluster Feature Selection (MCFS) are each described in detail in Section 2.6.1.

MDSC is used as the underlying clustering algorithm in the main experimental section and subsequently ACSC and CEDAS [87] are evaluated with the dynamic feature mask.

5.1.2 Creating and Maintaining the Mask

Assuming a window of $bufferSize$ points in d dimensions, unsupervised feature selection is performed on this window and the top n features are extracted. This subset of features is referred to as the the Current Features (CF). Formally: $CF = \{cf_1, \ldots, cf_n\}$, where $\{cf_i \in \mathbb{N}^+ \mid cf_i \leq d\}$. This subset of features CF is used to create a binary mask, referred to as the Current Mask (CM). Here, $CM = \{cm_1, \ldots, cm_d\}$, where:

$$cm_i = \begin{cases} 1: & \text{if } i \in CF \\ 0: & \text{otherwise} \end{cases} \tag{5.1}$$

Note here that $|CM| = d$ and the n features in CF will be represented as 1 and the others as 0.

These two sets (CF and CM) are calculated at each β window and are used to update a persistent vector of the feature values (FV). The feature values are the perceived importance or relevance of each feature at any given time. $FV = \{fv_1, \ldots, fv_d\}$, where $\{fv_i \in \mathbb{R} \mid 0 \leq fv_i \leq 1\}$. FV is updated after each window using the values in CM, as follows:

$$fv_i = \frac{fv_i + cm_i}{2} \tag{5.2}$$

It is the rolling average of each feature's importance (according the CM at each window) as the stream progresses. Finally, the DFM is updated based on the feature's importance in FV and a pre-defined threshold ν. $DFM = \{dfm_1, \ldots, dfm_d\}$, where:

$$dfm_i = \begin{cases} 1: & \text{if } fv_i \geq \nu \\ 0: & \text{otherwise} \end{cases} \tag{5.3}$$

The ν threshold ($\nu \in \mathbb{R} \mid 0 \leq \nu \leq 1$) dictates the length of time a feature is considered relevant if it is no longer selected in the top n features. A high threshold makes it harder for a new feature to be considered and also makes it easier to be discarded. A lower threshold maintains a previously important feature's relevance in DFM even if it is no longer selected.

After each β-window, a snapshot of the feature values is stored offline. This can be used to quickly examine a feature's importance over time.

5.1.3 Applying the Mask

To initialise the process, $bufferSize$ points are read into the buffer, the DFM is created and then clustering is performed using this mask. After initialisation we have a DFM and a set of clusters. Incoming points are clustered using the DFM. In density clustering algorithms, clusters are typically composed of micro-clusters and an incoming point is assigned to the most appropriate micro-cluster. This is determined by the distance from the point p to a micro-cluster m's center c provided that this distance is less than r, the radius of the micro-cluster. The distance is measured along *each* feature f_i in p to the center of m. With the DFM, we are only interested in taking the distance along the relevant unmasked features.

Center c and radius r for a micro-cluster m (Eqn. (3.1) and Eqn. (3.2), respectively) require the Linear Sum (LS) and Squared Sum (SS) of the N points described by m. To recap, m describes N points ($X_j \in m$, $j = \{1, ..., N\}$), and X_j is composed of d features X_{ji}, $i = \{1, ..., d\}$, where j is the instance and i the feature. The Linear Sum of feature i is calculated as $LS_i = \frac{1}{N} \sum_{j=1}^{N} X_{ji}$ and the Squared Sum of the feature is $SS_i = \frac{1}{N} \sum_{j=1}^{N} {X_{ji}}^2$. To apply the mask, each feature is multiplied by its counterpart in the binary DFM and only the non-zero features are considered.

$$\hat{LS_i} = \frac{1}{N} \sum_{j=1}^{N} X_{ji} dfm_i \tag{5.4}$$

$$\hat{SS_i} = \frac{1}{N} \sum_{j=1}^{N} {X_{ji}}^2 dfm_i \tag{5.5}$$

For the incoming point p do the same:

$$\hat{p_i} = p_i dfm_i \tag{5.6}$$

The process of creating and maintaining the DFM, and clustering using the DFM is outlined in Algorithm 13.

Algorithm 13 Clustering with a Dynamic Feature Mask

Input : Clustering Algorithm $Clust$, Feature Selector $Select$,
counter $counter$, incoming point p, $bufferSize$

1: **if** <$counter$ mod $bufferSize$ == 0> **then**
2: Current Features (CF) $\leftarrow$ $Select$ ($buffer$)
3: Use CF to generate Current Mask (CM). (Eqn. (5.1))
4: Use CM to update Feature Values(FV). (Eqn. (5.2))
5: Use FV to update Feature Mask (DFM) (Eqn. (5.3))
6: Clear buffer
7: Store latest FV off-line
8: **end if**
9: Apply DFM to p (Eqn. (5.6))
10: **for** <each cluster C> **do**
11: Apply DFM to C (Eqns. (5.4) & (5.5))
12: **end for**
13: $Clust(p)$
14: Add copy of p to buffer
15: $counter$ ++
16: Read next point

Table 5.1: Description of datasets used in experiments

Dataset	Classes	Features	Examples	Type
MNIST	5	784	26,000	Image
COIL-20	20	1,024	1,440	Image
NewsGroup	7	60,881	14,000	Text
TDT-2	30	36,771	9,494	Text

5.2 Experimental Study

This section presents experimental results using the proposed method. It is evaluated on four high-dimensional data streams exhibiting feature drift, feature evolution, concept drift and concept evolution. The previously described metrics (Purity, F-Measure, and Rand Index) are used to evaluate the method.

5.2.1 Datasets

Here, the four datasets used to evaluate the proposed method are described: two image-streams and two text-streams. An overview is presented in Table 5.1.

The popular MNIST benchmark dataset[1] is taken and converted to a stream in order to simulate concept and feature drift. MNIST consists of 26,000 grey scale,

[1]http://yann.lecun.com/exdb/mnist/

Table 5.2: Make-up of MNIST Stream.

	0	1	2	3	4
01-04k	2,000	2,000	0	0	0
05-08k	1,332	1,332	1,336	0	0
09-10k	500	500	500	500	0
11 -14k	800	800	800	800	8000
15-18k	0	1,000	1,000	1,000	1,000
19-22k	0	0	1,332	1,332	1,336
23-26k	0	0	0	2,000	2,000

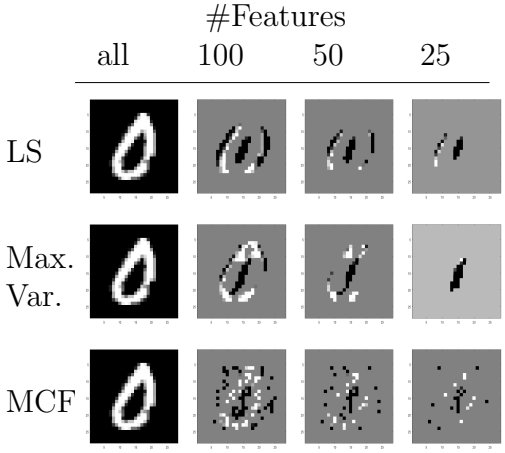
handwritten digits. To convert to a stream, five classes are taken from the original dataset (digits 0-4) and introduced to the stream in a sequential order. The first 4,000 instances contain images of digits 0 and 1 (shuffled), the following 4,000 points contain images of digits 0, 1 and 2, and so on. The makeup of the stream is presented in Table 5.2. The features in this stream are pixels and the discriminatory power of a pixel will change over the course of the stream. For example, the subset of pixels which can best describe digits 0 and 1 might not be useful to describe the digits which appear later in the stream. The second image stream is the Columbia Object Image Library (COIL-20) dataset, which consists of 1,440 normalised grey scale images. Images of 20 household objects are taken at different angles. It is converted to a stream by reading the data in sequential order. The different image classes arrive in sequence (class 1, then class 2 etc.), simulating concept evolution and feature drift.

The mask is further evaluated on two benchmark text-streams: 20Newsgroups and the Topic Detection and Tracking Corpus (TDT-2). 20Newsgroups[2] is a collection of 14,000 documents separated into 7 topics and further divided into 20 different sub-topics. Some of these sub-topics are very closely related (for example, PC-Hardware and Mac-Hardware), so in the evaluation the root of the topic is taken as the ground truth, this gives 7 topics: 'Alternative', 'Computers', 'Miscellaneous', 'Recreation', 'Science', 'Society', and 'Talk'.

The datasets are split into chunks of 1,000 and each chunk is shuffled in order to remove any bias (for example, a window containing only documents belonging to a single topic). The data is shuffled chunk-by-chunk in order to maintain the progression of topics in the stream and each chunk contains between 2 and 5 topics.

[2]http://qwone.com/ jason/20Newsgroups

Table 5.3: Features Selected on MNIST Stream



As a pre-processing step, stop-words ('a', 'the', 'and', etc.) are removed from the data-set giving a feature space of 60,881 words. This data stream is referred to as Newsgroup (as opposed to 20Newsgroups).

TDT-2 [64] consists of data taken from 6 sources; 2 news-wires, 2 radio, and 2 television programmes. TDT-2 consists of 2 months of reports and is often used as the training set in text-classification tasks. It consists of 9,494 documents divided into 30 topics. Again, stop words are removed, the data-set is divided into chunks of 1,000 and shuffled to remove any bias. A stream is simulated by reading the data in sequential order.

5.2.2 Evaluation

For each data-stream in this section:

- Three different selection methods for creating and maintaining the DFM are evaluated: Laplacian Score, Minimum Variance, and Multi-Cluster Feature Selection.
- The performance of a clustering algorithm with the DFM is evaluated, the performance without a mask is evaluated, and finally the performance with a static mask. The static mask performs feature selection on the first window and is never updated as the stream progresses.

On the MNIST stream, a β-window of 1,000 points is used and three unsupervised methods are evaluated to maintain the mask. The first window contains two classes: digits 0 and 1. Feature-subsets of different sizes (the top 100 features, the top 50, and the top 25) are evaluated and the original images are recreated using the selected features. This is illustrated in Table 5.3.

Table 5.4: Average time required (secs.) for feature selection on different window sizes

Method	1k	2k	5k
Var	0.01 (.005)	0.02 (001)	0.05 (.003)
LS	2.71 (.01)	11.6 (.03)	76.3 (.29)
MCFS	0.63 (.01)	1.34 (.006)	3.75 (.04)

Table 5.5: Comparative performance of different selection methods on MNIST using Purity (P), F-Score (F), and Rand-Index (R)

#features	LS			MCFS			Var		
	P	F	R	P	F	R	P	F	R
100	0.84	0.74	0.79	**0.87**	**0.78**	**0.82**	0.78	0.74	0.79
50	0.81	0.74	0.78	0.85	0.78	0.81	0.78	0.74	0.79
25	0.77	0.72	0.78	0.77	0.70	0.73	0.76	0.72	0.78

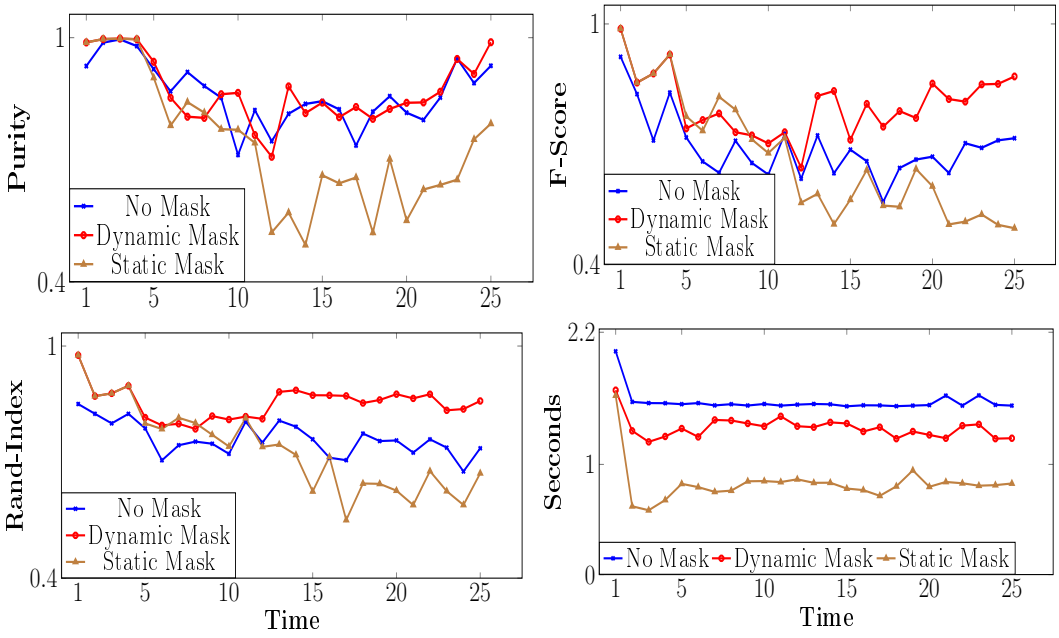


Figure 5.1: Comparative improvement in clustering performance on the MNIST stream

Also reported is the time each algorithm requires in Table 5.4. LS and Var appear to select similar features but LS takes substantially longer. The clustering performance using a DFM with different selectors and feature sizes is presented in Table 5.5. MCFS with 100 features creates the best DFM across the three metrics. The comparative improvement in the underlying clustering algorithm (MDSC) is illustrated in Fig. 5.1. The DFM improves clustering on all three metrics and also

Table 5.6: Features Selected on COIL-20 Stream

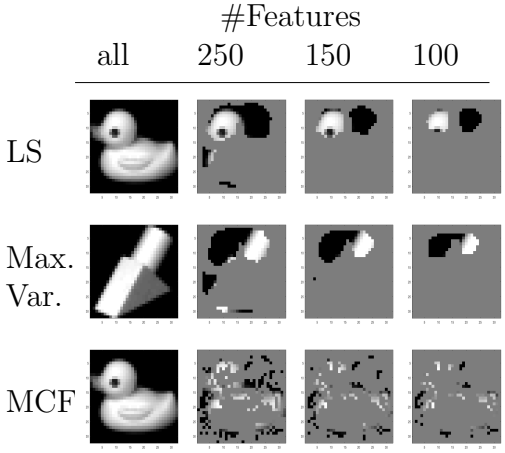


Table 5.7: Comparative performance of different selection methods for creating the DFM on COIL-20 stream

#features	LS			MCFS			Var		
	P	F	R	P	F	R	P	F	R
250	0.86	0.69	0.73	**0.94**	**0.87**	**0.86**	0.86	0.77	0.75
150	0.84	0.68	0.73	0.92	0.87	0.86	0.86	0.77	0.73
100	0.87	0.75	0.75	0.89	0.87	0.84	0.86	0.74	0.72

requires less time. This is because fewer pair-wise distance calculations are required. Without a mask, measurements are taken along each of the dimensions but with a mask only the important features are considered. This time measurement includes the time it takes to perform feature selection. The static mask is fastest; it requires fewer pairwise calculations and does not perform feature selection after the first window. Although it is faster, the performance suffers and it is better to use no mask at all rather than a static mask.

COIL-20 contains fewer samples than MNIST so, a β-window of 100 instances is used. However, it has a larger number of dimensions and larger feature-subsets (top 250, 150 and 100) are taken. The first window contains two classes and the features selected by each FS method are displayed in Table 5.6.

As in MNIST, LS and Var. appear to select similar features. The average performance using each FS method (with different feature subset sizes) over the entire stream is presented in Table 5.7. Again, MCFS creates a better mask than LS and Var. The comparative performance of the DFM with a static mask and no mask is presented in Table 5.8. Clustering using the DFM returns a better performance than clustering without a mask. Clustering using a static mask returns

Table 5.8: Average Clustering Performance on COIL-20 Stream

	Purity	F-Score	Rand Index
Dynamic Mask	**0.94**	**0.87**	**0.86**
Static Mask	0.91	0.74	0.79
No Mask	0.92	0.81	0.81

Table 5.9: Comparative performance of different selection methods for creating the DFM on NewsGroup stream

#features	LS			MCFS			Var		
	P	F	R	P	F	R	P	F	R
500	0.93	0.74	0.64	0.91	0.65	0.48	0.90	0.77	0.76
250	0.94	0.76	0.66	0.86	0.74	0.61	0.88	0.79	0.78
150	0.94	0.76	0.67	0.87	0.75	0.65	**0.93**	**0.82**	**0.81**

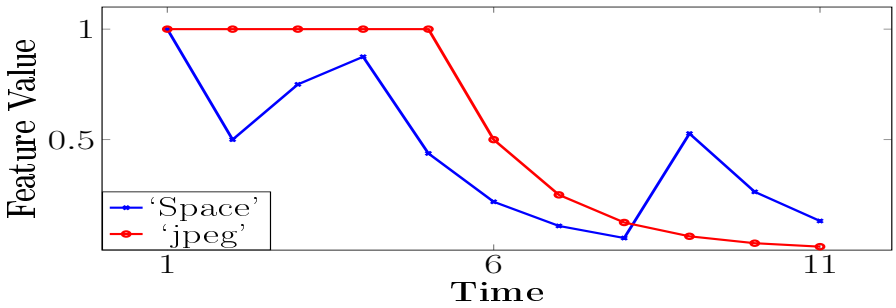


Figure 5.2: Tracking feature-drift on two words in the NewsGroup stream.

the worst performance out of the three, again.

The text-streams have much higher dimensionality than the grey-scale images. So, larger feature subsets (up to 500 features) are taken with a window-size of 1,000. Over the entire stream, the Maximum Variance selection method creates the best mask as can be seen in Table 5.9. The first window contains two topics: 'Alternative' and 'Computers'. The top 5 features selected by Maximum Variance are: {*jpeg, image, graphics, Jesus, God*}. Using the Feature Values vector, which is updated after each window, we can track the importance of a word as it changes over time. As an illustrative example, two words are taken from the first window - 'jpeg' and 'space'. Their perceived importance over the course of the stream is displayed in Fig. 5.2. 'jpeg' is considered important for the first five windows but begins to loose importance as the 'computers' topic disappears from the stream. It is never selected again and by the end of the stream its perceived importance is zero. 'Space' is also selected in the context of computing and its importance drops as the 'computing' topic disappears from the stream. However, the word becomes relevant

Table 5.10: Average Clustering Performance on NewsGroup Stream

	Purity	F-Score	Rand Index
Dynamic Mask	**0.93**	**0.82**	**0.81**
Static Mask	0.85	0.76	0.72
No Mask	0	0	0

Table 5.11: Comparative performance of different selection methods for creating the DFM on TDT-2 stream

#features	LS			MCFS			Var		
	P	F	R	P	F	R	P	F	R
500	0.77	0.60	0.61	0.80	0.62	0.61	0.72	0.63	0.62
250	0.77	0.60	0.58	0.79	0.63	0.62	**0.83**	**0.63**	**0.61**
150	0.81	0.59	0.58	0.77	0.62	0.61	0.83	0.62	0.58

Table 5.12: Average Clustering Performance on TDT-2 Stream

	Purity	F-Score	Rand Index
Dynamic Mask	**0.83**	**0.62**	**0.61**
Static Mask	0.72	0.61	0.60
No Mask	0	0	0

again later in the stream, in a different context; 'space' is once again selected when the 'Science' topic is present in the stream. 'Space' is selected along side features such as 'satellite', 'NASA', and so on.

Without a mask, no clustering solution is found. This is likely because of the high dimensionality (>60,000). However, with a static mask of 150 features, a solution is returned. Clustering performance is further improved using a dynamic mask, which is summarised in Table 5.10.

On the TDT-2 stream, the Maximum Variance selection method provides the best DFM. The comparative performance with the other two selection methods is displayed in Table 5.11, and the clustering improvement in Table 5.12.

The results across each data stream are summarised in Table 5.13. On each of the four data-streams, on all three metrics, the MDSC algorithm is improved using the proposed DFM.

In all of the previous experiments described MDSC was used as the underlying clustering algorithm to test the proposed DFM.

It is further evaluated on two other density based cluster algorithms; ACSC and CEDAS [87]. On each dataset, the best selection method discovered in previous

Table 5.13: Performance of Dynamic Mask with MDSC

Data-Stream	No Mask			Static Mask			Dynamic Mask		
	P	F	R	P	F	R	P	F	R
MNIST	0.86	0.69	0.75	0.73	0.66	0.73	**0.87**	**0.78**	**0.82**
COIL	0.92	0.81	0.81	0.91	0.74	0.79	**0.94**	**0.87**	**0.86**
NewsGroup	0	0	0	0.85	0.76	0.72	**0.93**	**0.82**	**0.81**
TDT-2	0	0	0	0.72	0.61	0.60	**0.83**	**0.62**	**0.61**

Table 5.14: Performance of Dynamic Mask with ACSC

Data-Stream	No Mask			Static Mask			Dynamic Mask		
	P	F	R	P	F	R	P	F	R
MNIST	0.89	0.69	0.75	0.75	0.65	0.73	**0.92**	**0.80**	**0.84**
COIL	0.86	0.79	0.74	0.85	0.74	0.72	**0.92**	**0.82**	**0.79**
NewsGroup	0	0	0	0.81	0.70	0.72	**0.90**	**0.82**	**0.81**
TDT-2	0	0	0	0.72	0.61	0.60	**0.87**	**0.60**	**0.61**

Table 5.15: Performance of Dynamic Mask with CEDAS

Data-Stream	No Mask			Static Mask			Dynamic Mask		
	P	F	R	P	F	R	P	F	R
MNIST	0	0	0	0.78	0.62	0.61	**0.91**	**0.64**	**0.73**
COIL	0.5	0.17	0.53	0.99	0.67	0.72	**0.99**	**0.69**	**0.75**
NewsGroup	0	0	0	0.84	0.65	0.68	**0.90**	**0.74**	**0.78**
TDT-2	0	0	0	0.74	0.58	0.57	**0.89**	**0.58**	**0.59**

experiments is used; MCFS with 100 and 250 features for MNIST and COIL, respectively, and Maximum Variance with 150 and 250 features for Newsgroup and TDT-2, respectively. The comparative results are displayed in Table 5.14 using ACSC, and Table 5.15 using CEDAS. On every stream, each of the underlying clustering algorithms is improved by the proposed DFM.

5.3 Sensitivity Analysis

In this section, the sensitivity and effect of the two parameters required to create and maintain the DFM are evaluated; the threshold value ν and the window size $bufferSize$.

Experiments are performed with the MNIST data stream using MCFS with 100 features as the selector. For illustrative clarity, 'Score' is used as the metric.

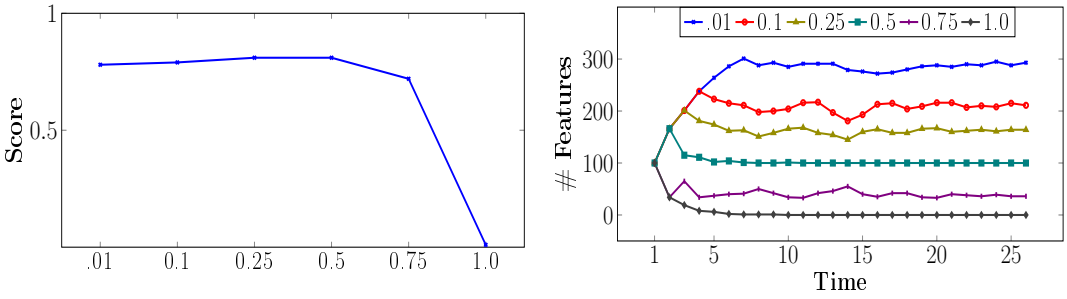


Figure 5.3: Sensitivity of ν with respect to clustering performance (left) and the effect of the parameter on the number of features considered in the clustering process (right).

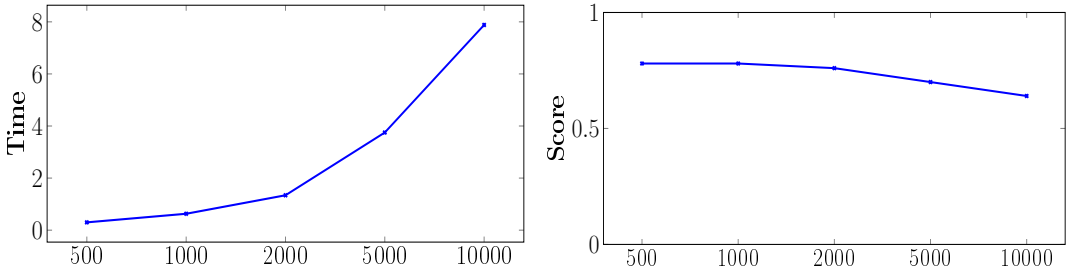


Figure 5.4: Time required to perform FS on different values for $bufferSize$ (left) and sensitivity of $bufferSize$ with respect to clustering performance (right).

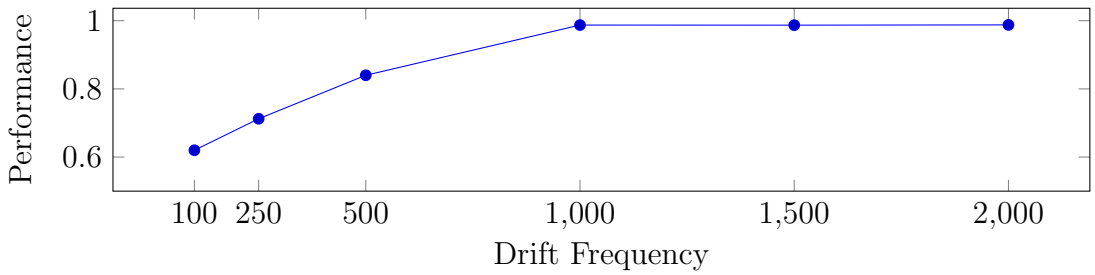


Figure 5.5: Sensitivity of $bufferSize$ to underlying concept drift

ν determines the length of time a feature remains relevant if it is no longer selected in top n features. It is a threshold for the Feature Values and determines which features are considered in the clustering process. Experiments use values in the range 0.1 to 1.0. The results are displayed in Fig. 5.3 (left). Clustering performance is stable with a slight drop after a value of 0.5. If the threshold is too high (1.0 in this example), the performance suffers dramatically. If the threshold is too high, no features are considered so the clustering process does not happen. In this case, clustering would only occur on features which have been selected in every window. This is perhaps unlikely in a dynamic stream. This is illustrated in Fig. 5.3

(right). Here, the number of features that are considered in the clustering process is displayed. The top 100 features are selected in each window and with a low threshold, previously important features remain relevant for a long time even if they are no longer being selected. This can be seen with a ν value of 0.01, approximately 300 features are considered as 'important' at each time-step. With a high threshold of 1.0, no features are considered important by the end of the stream. Using a value of 0.5, the number of selected features remains at roughly 100. For each experiment described in this chapter, a ν value of 0.5 is used.

The parameter $bufferSize$ determines the number of points which should be collected in the buffer before feature selection is performed and the DFM is updated, i.e. the size of the β-window. First, the time it takes to perform FS on different β-windows is measured. Window-sizes from 500 to 10,000 are examined and measured in seconds. The results are displayed in Fig. 5.4. It can be seen that the relationship between time and the β-window is not quite linear and it is more efficient to use smaller values for $bufferSize$. This is confirmed when measuring the clustering performance using the different window sizes. The score decreases as $bufferSize$ increases. In this chapter, a value of 1,000 for $bufferSize$ is used in all experiments except for COIL-20 which is comparatively small and a value of 100 was used instead.

The parameter $bufferSize$ is evaluated in terms of how sensitive it is to the rate of concept drift. In this experiment, $bufferSize$ is fixed to $1,000$, so, the relevant subset of features are evaluated (and the mask updated) every 1000 points. With $bufferSize$ fixed, the rate at which the concepts change is altered. Again, the MNIST stream is used to evaluate this. Concept drift is simulated every 100 points, then every 250 points, then every 500, and so on. The results are displayed in Figure 5.5. It can be seen that performance suffers if change occurs more frequently than the mask is updated . If $bufferSize$ is roughly the same size as the rate of change (or if the mask is updated more often than change occurs) then performance remains good and stable. The sensitivity of this parameter to underlying change provides further evidence that it is better to keep this value low and update the mask as frequently as possible.

5.4 Summary

This chapter presents a Dynamic Feature Mask (DFM) for unsupervised dynamic feature selection in non-stationary data-streams. Redundant features are masked and clustering is performed along unmasked, relevant features. If a feature's perceived importance changes, the mask is updated accordingly - previously unimpor-

tant features can be unmasked and features which loose relevance become masked. The method is proposed to address two challenges in data-stream clustering: 1) feature drift - a change at the feature level in a stream, and 2) the problem of clustering high-dimensional streams where the curse of dimensionality renders distance measurements, and the concepts of 'density', difficult.

The proposed method is algorithm-independent and can be used with any existing density based clustering algorithm. Two density-based clustering algorithms have been proposed in previous chapters and neither have a mechanism to deal with feature drift or with very high-dimensionality.

The proposed method was evaluated on three density based clustering algorithms (MDSC, CEDAS, and ACSC) across four high-dimensional streams; two text streams and two image streams. In each case, the proposed DFM improves clustering performance and furthermore, reduces the processing time required by the underlying algorithm.

An unsupervised feature selection method is required to create and maintain the DFM and three existing methods were evaluated: Laplacian Score, Multi-Cluster Feature Selection, and Maximum Variance. Experimental results suggest that on the lower dimensional ($\approx 1,000$ dimensions) streams, MCFS is the best selector for the mask. On the higher dimensional text streams (up to 60,000 dimensions), the Maximum Variance method selects the best features to maintain the mask. The Laplacian Score did not return the best features on any stream and was shown to require considerably more time than the other two methods.

On each stream, the DFM was compared with a static feature mask. In the static case, the mask is created on one window at the beginning of the stream and is never updated. The dynamic mask performs better on each stream. On the higher dimensional streams, the static mask is preferable to no mask (without a mask the clustering algorithms could not return a solution at all) but on the lower dimensional streams it is preferable to use no mask rather than a static mask.

Chapter 6

Clustering and Classification Ensemble

Ensemble techniques have been shown to be a powerful method for recognising and reacting to change in non-stationary data. However, the bulk of research into dynamic classification with ensembles assume that the true class label of each incoming point is available or easily obtained. This is unrealistic in most practical applications especially in high-velocity streams where manually labelling each point is not practical. To address this challenge, this chapter introduces a framework incorporating a stream clustering algorithm and an ensemble of one-class classifiers with active learning. The framework aims to exploit the intuitive relationship between clusters and one-class classifiers (OCC). The framework: Clustering and One-Class Classification Ensemble Learning (COCEL) is intended to cope with a small training set (or no training set) and improve with experience, self-modifying its internal state in order to cope with underlying changes in the data-stream.

A set of OCCs is maintained in an ensemble, each responsible for identifying a particular class (or part of a class if, for example, the class has a multi modal distribution). Data points arrive online and the ensemble attempts to classify the point. If the point is unrecognised by the ensemble (concept evolution, concept drift, or noise/outlier) the point is passed to an online stream-clustering algorithm. When a new cluster is discovered, representative samples from the cluster are passed to an expert for labelling. This label is propagated to all points in the cluster and a new classifier is trained on these points and added to the ensemble.

The proposed framework can identify and react to concept evolution, concept drift and hugely reduces the number of required labels, especially in times of stability when updates are not needed. The framework signals change in the stream and potentially makes it easier for the user to gauge exactly what kind of change is

occurring.

Cluster-and-classifier ensembles have been previously proposed in the literature, especially in a scarcity of labels environment (Section 2.4). However, none in the literature use a stream-clustering algorithm. Typically, k-means is used along with a sliding window [88, 125, 138, 139, 82, 178, 121, 122, 177]. This is at odds with recent approaches to stream-clustering where k-means is very rarely used because 1) it requires more than a single pass of the data, 2) only spherical clusters can be found, 3) k needs to be specified and 4) it is typically performed off-line. The method proposed in this chapter uses MDSC coupled with an ensemble of OCCs.

Generally a one-class classifier ϕ takes the form:

$$(m_\alpha, x_i) \geq \theta = \begin{cases} 1: & x_i \text{ is a target} \\ 0: & x_i \text{ is an outlier} \end{cases} \tag{6.1}$$

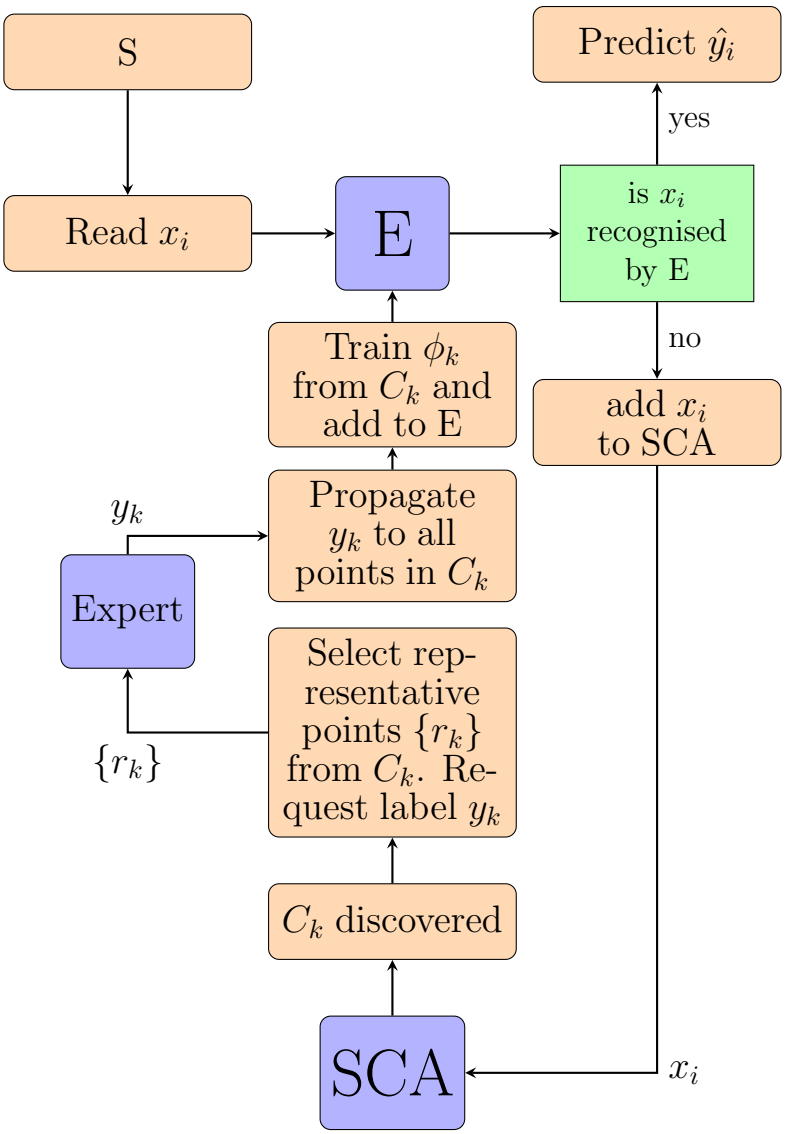
with a model m with parameters α and a threshold $\theta \in R$. There has been much research in one-class classifiers. Many options exist and certainly the "no free lunch" theorem applies. Some OCCs, for example, Minimum Spanning Trees have been shown to work well in high dimensions with relatively few training samples. Others, in the family of density based OCCs require a larger number of training samples.

Considering that, in the proposed framework, classifiers are trained automatically, it is desirable to have as few sensitive parameters as possible. Traditional parameter tuning techniques (cross-validation, etc.) might not be feasible in a data-stream with time and memory constraints. Or perhaps, not *every* classifier in the ensemble could be afforded long training times but some could – in the case of heterogeneous ensembles. An OCC based on micro-classifiers is described in this chapter. Micro-classifiers are used as they essentially come "for free" with a density based clustering algorithm. A micro-classifier is a lazy-learning micro-cluster, there is no training phase and the testing phase (when a point arrives to be evaluated) is faster than other lazy learning techniques (for example k-nearest neighbours) because a group of similar points have been summarised into a single micro-classifier. The micro-classifier described is an extension to the micro-clusters discovered by MDSC: the parameters are discovered adaptively, they are time-weighted and can act as a dynamic OCC.

In summary, the main contribution of this chapter are:

- A novel framework for classifying dynamic data streams in a scarcity of labels is proposed.
- A novel dynamic One Class Classifier is described.

Figure 6.1: COCEL Framework



6.1 Clustering and One-Class Classification Ensemble Learning

In the proposed framework, an assumption is made that either a small labelled training set is initially available or no training set is available at all. If a training set is available, it is split into each constituent class $Y = \{y_1, \ldots, y_n\}$ and clustering is performed on each class y_i. For each discovered cluster c_i, a classifier ϕ_i is trained using the clustered points. This classifier is added to the ensemble E, where $E = \{\phi_1, \ldots, \phi_m\}$. Note, $|E| \geq |Y|$, for example more than one cluster might be found in a particular class. A classifier ϕ_i is described as $\phi_i = [m_i, y_i, t_i]$, where m_i is the trained model, y_i the associated class label and t_i, a time-stamp.

Points arriving from stream S are passed to E and, if recognised by at least one classifier $\phi_i \in E$, a prediction $\hat{y}_i$ is made, and the time stamp of the classifier which made the prediction is updated, $\phi_{t_i} = T$. In order to regulate the size of the ensemble, classifiers age and are removed if they are no longer making predictions. If a classifier's time-stamp t is greater than $T - \lambda$, the classifier is considered no longer useful and is removed from E.

If the incoming point is unrecognised by all classifiers in the ensemble, it is passed to an online stream clustering algorithm SCA. A point could be unrecognised because it is simply a noise or outlier point or it could signal change in the form of evolution or drift. If the point does represent change, then we would expect more, similar points to be added to SCA and clusters to form. If a cluster c_k is discovered in SCA and $|c_k| > minClusterSize$, a subset of representative points $\{r_k\}$ are selected from c_k and passed to a human expert for labelling. This label; y_k, is propagated to all points in c_k and a new one-class classifier ϕ_k is trained on the points in c_k and is added to E. If no initial training set is available, E initialises as empty and incoming points are passed directly to SCA until clusters are discovered and their resultant classifiers added to E. The proposed framework is outlined in Fig. 6.1.

It is clear that there are two major considerations to be made; the choice of stream clustering algorithm and the choice of one-class classifier (or heterogeneous combination thereof). In this chapter MDSC is used as the underlying clustering algorithm with a homogeneous ensemble of OCCs. A time-weighted micro-classifier is proposed as the OCC.

6.2 Time Weighted Micro-Classifier

Previously, an OCC was defined as $\phi_i = [m_i, y_i, t_i]$. For a micro-classifier $m\phi_i$, the model m consists of a center cen and radius r so $m\phi_i = [cen_i, r_i, y_i, t_i]$. The time-stamp t_i records the most recent time at which $m\phi_i$ made a prediction used by the ensemble. An incoming point x_i is recognised if it falls within the hypersphere of $m\phi_i$, *i.e.* if the Euclidean distance from x_i to the centre of $m\phi_i$ ($m\phi_{i_{cen}}$) is less than the radius of $m\phi_i$ ($m\phi_{i_r}$). Formally;

$$m\phi_i(x_i) = \begin{cases} m\phi_{iy}, & \text{if } d \leq m\phi_{ir} \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

where $d = distance(x_i, m\phi_{i_{cen}})$. If x_i actually is a target, d is recorded as a measure of confidence in the micro-classifier's prediction. This confidence is time-weighted

using an exponential decay function; e^{wt} (where w is a constant and t is $m\phi$'s time-stamp). The ensemble will trust recent micro-classifier's predictions more than older ones.

The two parameters required by the classifier are r and *cen*. If the underlying clustering algorithm is density based (uses micro-clusters) then these two parameters are taken directly from their respective micro-clusters. With MDSC, r is adaptive and local to each micro-classifier (though, the same for each micro-cluster in the same macro-cluster) creating micro-classifiers with varying radii.

The difference between the classifier described and other micro-classifiers is the addition of the time-weight to the classifier's confidence and also how a classification decision is made. Others ([125, 138, 139]) use a nearest neighbour method whereby a point is classified by its proximity to k micro-clusters discovered in the training set. Others ([122, 177]) use a global maximal-distance allowed between a point and its neighbouring micro-classifier before a classification decision is made (unlike the local distance r used here) and finally the fact that classifiers can be created in real time without windowing differentiates this method from others.

6.2.1 Time Weighted Micro-Classifiers in the COCEL Framework

In the proposed framework, for each newly discovered cluster c_k a new OCC ϕ_k is created. For most OCCs this is a one-to-one relationship, for example if a Support Vector Machine is used as the OCC, there will be one SVM for each discovered cluster. With micro-classifiers, for each c_k there will be n corresponding classifiers *i.e.* $\phi_k = \{m\phi_1, \ldots, m\phi_n\}$.

Micro-classifiers can overlap and an incoming point x_i can sometimes be recognised by more than one classifier. This is especially true in streams with a lot of change and in data where concepts are not clearly linear-separable. The final decision of the ensemble is the label associated with the most confident micro-classifier. When the ensemble makes a prediction, the time-stamp of the most confident micro-classifier is updated.

6.3 Active Learning Process

Typically, the challenge in active learning is to select the most representative and informative samples from a set of unlabelled points. In the proposed framework this challenge is made easier by the clustering process. Points which are recognised by the

Table 6.1: Training Data Used in Experiments

Dataset	Classes (in training set)	Samples (training)
$4CR$	4 (4)	144,000 (500)
$5C$	5 (2)	200,000 (500)
Network	5 (1)	494,000 (500)
Forest	7 (2)	580,000 (500)

ensemble are not required to be labelled, furthermore points which are unrecognised but are noise will not be clustered so will never need to be labelled.

This leaves a much smaller subset of unrecognised, useful points which can potentially be actively labelled. These points have been clustered so occupy the same area of the decision space. Based on this, and the assumption that data from the same class are close in the decision space, the process is simplified to selecting in-cluster samples.

In the subsequent experiments, n samples are selected randomly from each newly discovered cluster. A label y is given to the cluster based on these n samples (the majority label is used if there is a conflict). Finally, y is propagated to all points in the cluster.

6.4 Experimental Study

6.4.1 Datasets

The framework is tested on four datasets; two synthetic and two real. The synthetic datasets are selected because they contain concept drift (real and virtual) and concept evolution. One exhibits sudden changes and the other a slower, gradual change. Two real data sets are selected which would be realistic applications of the COCEL framework. The first is the previously described Network Intrusion data-stream. Network traffic arrives at high speed and if malice is suspected, network requests might need to be examined by a human expert, however it is not realistic to label each incoming point. The second stream is the Forest Cover stream. This dataset uses cartographic variables to describe seven species of tree. In this situation it is easy to imagine that the cartographic data is easy to gather (x_i) but gathering the labels (y_i) is expensive and labour intensive. In each case it is assumed that there is a small labelled training set available. The size of each data-stream along with the details of the training set are presented in Table 6.1.

For illustrative purposes the performance of COCEL is compared with a static

ensemble. A static ensemble uses the training set but is never updated throughout the stream.

There are no comparative active-learning ensembles in which to perform a like-for-like comparison. Instead, the framework is compared with two state-of-the art fully supervised ensembles. OzaBoostAdwin [149] uses an ensemble of Hoeffding trees, trained using boosting with adaptive windowing (ADWIN) to remove out-dated and under-performing ensemble members (outlined in Section 2.4). Adaptive Random Forest (ARF) [72] is a data-stream adaptation of the original Random Forest algorithm, an ensemble of decision trees trained using bagging (outlined in Section 2.4, Algorithm 5). Both of the supervised ensembles are evaluated using the Massive Online Analysis (MOA) [24] open source software. For context, an active learning algorithm for data streams is included. This is not an ensemble method and does not deal with drift in the same manner. The Active Classifier [209] uses a single base model. Requests for labels are made if the model's confidence in a prediction is below a certain threshold. If drift is suspected, a new model is trained in parallel. If drift is confirmed then the new model replaces the old. This algorithm was also tested in MOA and is outlined in Section 2.4, Algorithm 6. With the Active Classifier, the number of labels that COCEL requires is used as the labelling budget. In the original paper, a VFDT is used as the classifier but in this comparison a Bayesian classifier is used as it achieves a better performance with such a low labelling budget.

6.4.2 Evaluation Metric

The performance of the algorithm is measured using prequential evaluation. Prequential (a contraction of predictive sequential) evaluation is the error computed over a sequence of points, i.e. the incoming test stream. It is the accumulated sum of the algorithm's accuracy. An incoming point x_i with a true label y_i is passed to the ensemble E and a prediction $\hat{y}_i$ is returned; $\hat{y}_i = E(x_i)$. The accuracy L for the prediction is calculated as:

$$L(y_i, \hat{y}_i) = \begin{cases} 1, & \text{if } y_i = \hat{y}_i \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

And the performance of the algorithm is the cumulative accuracy over the entire test stream:

$$Accuracy = \frac{1}{i} \sum_{i=1}^{\infty} L(y_i, \hat{y}_i)) \tag{6.4}$$

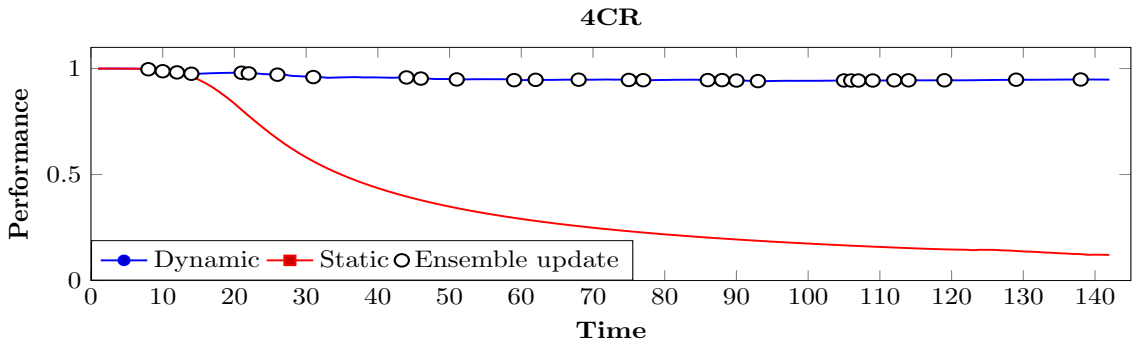


Figure 6.2: Performance of COCEL on 4CR Data-Stream

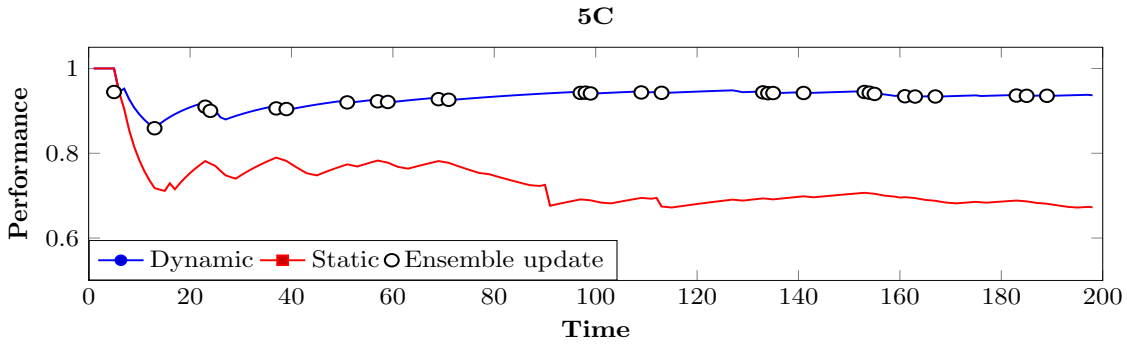


Figure 6.3: Performance of COCEL on 5C Data-Stream

6.5 Evaluation

6.5.1 Comparative Performance

The first 500 instances in each stream are taken as the training set (the effect of this parameter is examined in a subsequent section).

The first synthetic stream is the 4CR data stream. It consists of 4, 2-dimensional Gaussian classes. The classes revolve anti-clockwise and move into positions previously occupied by a different class. This change is gradual, beginning as virtual drift (a change in $P(x)$) but leading to real drift (a change in $P(y|x)$). In the training set, 4 clusters are discovered, generating 186 micro-classifiers in the ensemble. Classifiers are removed if they have not made a prediction in over 1,000 incoming instances and clusters with a minimum size of 10 are promoted to the ensemble. The performance of the framework is presented in Fig. 6.2. The framework maintains a high classification rate despite the underlying change. The static ensemble's performance begins to degrade immediately and never recovers. The ensemble was updated with new classifiers 29 times. These are represented as circles at the time when the update happens. COCEL achieves an accuracy of 94%, and required 587 labels (including the training set), 0.004% of the entire stream.

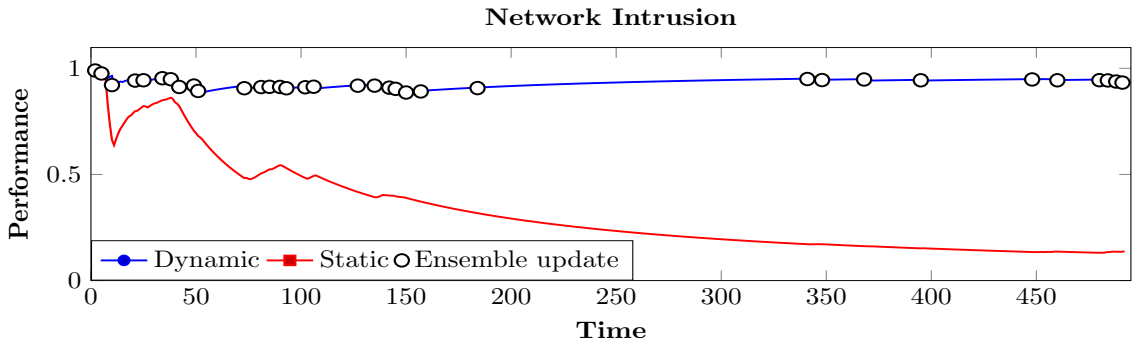


Figure 6.4: Performance of COCEL on Network Intrusion Data-Stream

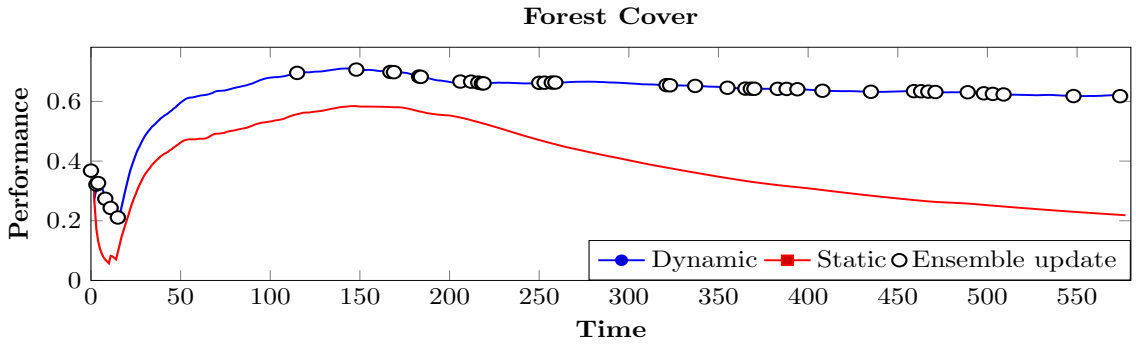


Figure 6.5: Performance of COCEL on Forest Cover Data-Stream

Next, a synthetic stream was created called $5C$. This stream is used to test the framework in a dynamic environment that contains sudden concept evolution along with concept drift (both virtual and real). The stream contains between 2 and 5 Gaussian classes. After a set drift interval, the centres and standard-deviation of each class distribution shift and there is a 50% chance of a class being removed or added to the stream, within bounds $[2, 5]$. The result is displayed in Fig. 6.3. An accuracy of 92% is achieved despite only requiring labels for 0.003% of the stream.

The performance of COCEL on the Network Intrusion data is displayed in Fig. 6.4. One class (the normal class) was discovered in the training set. One cluster with 326 micro-clusters initialised the ensemble. The remaining 4 drifting classes were discovered incrementally with active learning. The algorithm achieves a classification performance of 95% and required 602 labels, 0.001% of the entire stream.

The performance in the Forest Cover data-stream is displayed in Fig. 6.5. In the training set, two of the seven classes were present. 4 clusters generated 184 micro-classifiers. The algorithm achieves 68% prequential accuracy with 0.001% of the stream labels.

Finally, the results of the unsupervised framework are compared with two state-of-the-art supervised stream classifiers. COCEL's accuracy is comparable with the

Table 6.2: Comparative Performance of COCEL

	COCEL		*Active*		*OzaBoost*		*ARF*	
	Acc.	(*Labels*)	*Acc.*	(*Labels*)	*Acc.*	(*Labels*)	*Acc.*	(*Labels*)
$4CR$	0.96	(0.004%)	0.25	(0.004%)	0.99	(100%)	0.99	(100%)
$5C$	0.91	(0.003%)	0.57	(0.003)%	0.95	(100%)	0.98	(100%)
Network	0.95	(0.001%)	0.95	(0.001%)	0.95	(100%)	0.97	(100%)
Forest	0.68	(0.001%)	0.56	(0.001%)	0.86	(100%)	0.89	(100%)

supervised algorithms on three of the four datasets despite requiring only a fraction of the labels. It performs favourably to its peer active-learning algorithm with the same label budget. The results are displayed in Table 6.2.

6.5.2 Sensitivity Analysis

This section presents an analysis of the parameters required within the proposed framework:

1. The number of initial training samples available to the algorithm before streaming begins
2. The minimum size a discovered cluster must be in order to train a new classifier
3. The number of random samples taken from a cluster to be labelled by an expert
4. λ, which determines how quickly classifiers age in the ensemble. A large value for λ means classifiers persist for longer, even if they are not making any predictions

The results are displayed in Fig 6.6. It can be observed that only λ is sensitive to small changes in value. If this value is too small, useful classifiers are disregarded too easily, conversely, if it is too large, out-dated classifiers persist and make out-dated predictions.

While the values for training samples and $minClusterSize$ do not affect the performance a great deal, a larger value for each is preferable. It is interesting to note that even without a training set, the algorithm can learn to recognise concepts within a stream. For example, in the Network Intrusion stream with five classes, the framework achieves 89% accuracy while requiring only 126 labels, 0.0002% of the stream. An initial training set of 1,000 points achieves 95% accuracy.

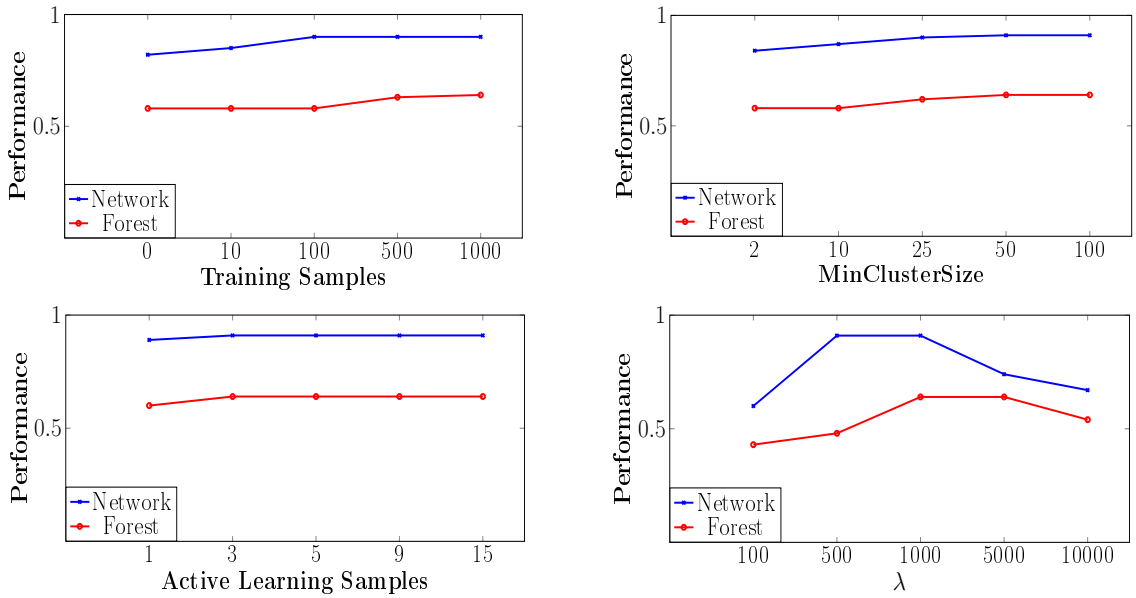


Figure 6.6: Sensitivity of Parameters in COCEL

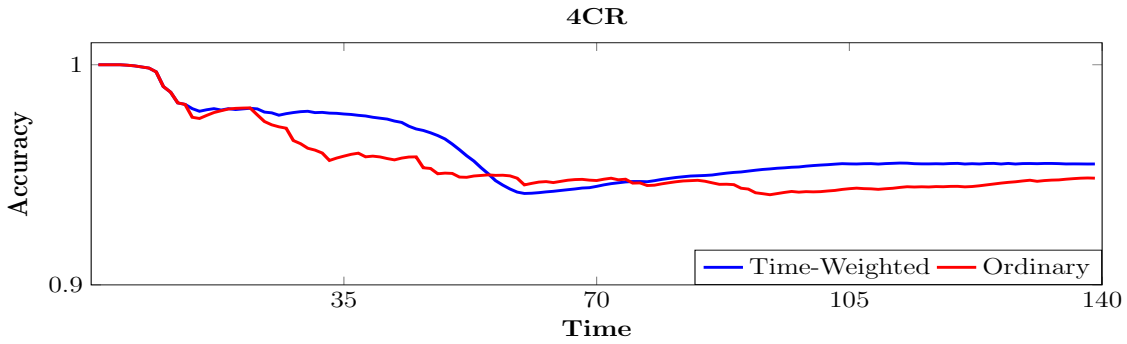


Figure 6.7: Comparative Performance of Time-Weighted classifier on 4CR data-Stream

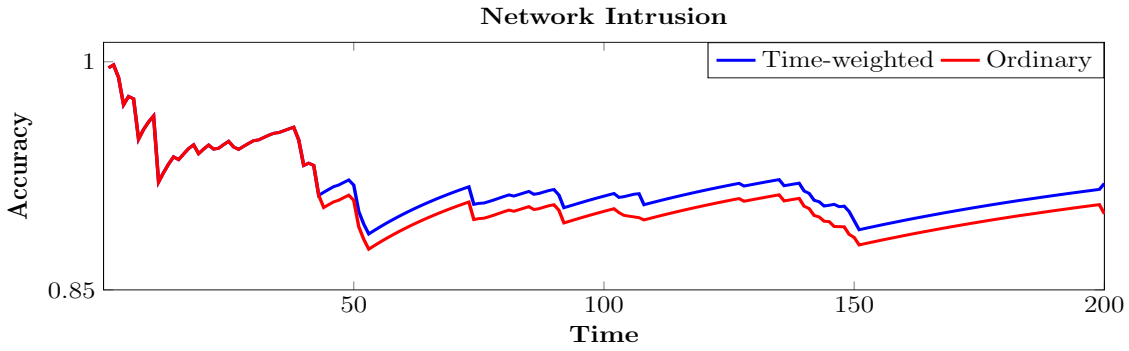


Figure 6.8: Comparative Performance of Time-Weighted classifier on Network Intrusion Data-Stream

Table 6.3: Label Selection Strategies in COCEL

Method	Network	Forest
Random	0.951	0.678
Closest	0.951	0.676
Farthest	0.951	0.677

6.5.3 Effect of Time-Weighting on a Micro-Classifier

In this section the effect of time-weighting the micro-classifiers is examined. The performance of an ensemble of un-weighted micro-classifiers is compared with an ensemble of time-weighted micro-classifiers on two streams: The 4CR synthetic stream and the Network Intrusion stream.

Un-weighted micro-clusters output the confidence of their prediction using only d, the distance from the incoming point x_i and $m\phi_{i_c}$; the centre of the micro-classifier (Eqn. (6.2)). In the weighted approach, this confidence is time-weighted by $\mathrm{e}^{\mathrm{w}t}$

The comparative performance is presented in Figures 6.7 and 6.8. It can be seen that the time-weighted classifier performs slightly better than its un-weighted counterpart.

6.5.4 In-cluster sample selection

In the previous experiments n random points are selected from each cluster and labelled. Two alternate approaches are evaluated; selecting n points closest to the centre of the cluster and then, n points which are furthest from the centre.

To evaluate the three approaches, the Network Intrusion Stream is used and with each newly discovered cluster, $n = 3$ samples are taken to label the cluster.

The results are displayed in Table 6.3. There is barely any difference in the three approaches. This is due to the purity of the discovered clusters. Out of the three approaches, random selection is used because it requires the least computation.

6.5.5 Evaluating Different Base Classifiers

In all experiments described so far, time-weighted micro-classifiers have been used as the ensemble members. In this section three other OCCs are evaluated, each in a homogeneous ensemble. The Support Vector Domain Description (SVDD) is a boundary OCC, Minimum Spanning Tree (MST) is density/distance-based, and Principal Component Analysis (PCA) is a reconstructive method.

When a new cluster c_k is discovered (and a label allocated) a classifier is trained.

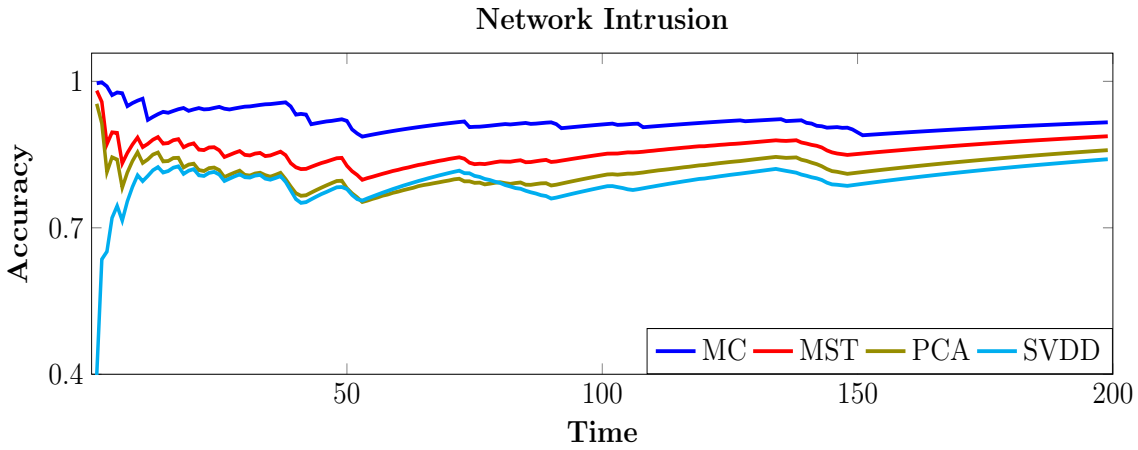


Figure 6.9: Comparative Performance of different base-learners on Network Intrusion Data-Stream. Time-Weighted Micro-Classifier (MC), Minimum Spanning Tree (MST), Principal Component Analysis (PCA), Support Vector Domain Description (SVDD)

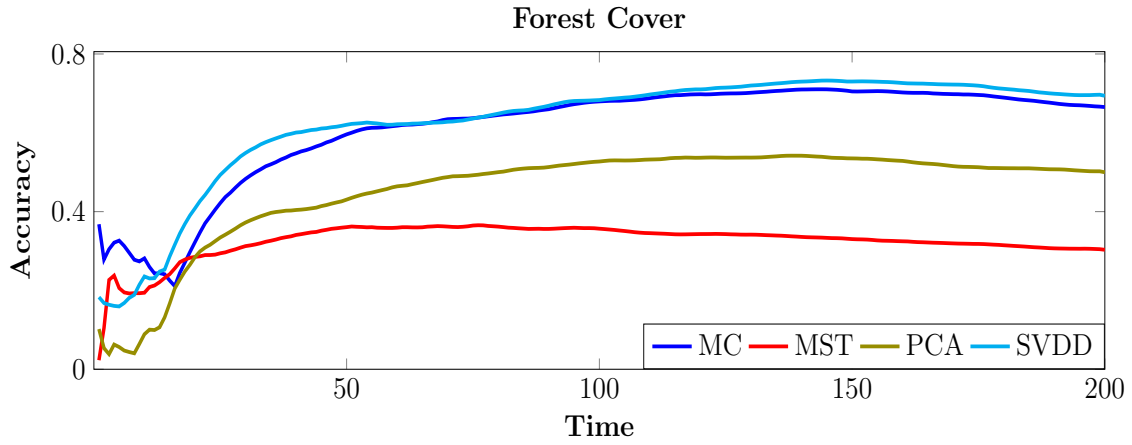


Figure 6.10: Comparative Performance of different base-learners on Forest Data-Stream. Time-Weighted Micro-Classifier (MC), Minimum Spanning Tree (MST), Principal Component Analysis (PCA), Support Vector Domain Description (SVDD)

With MST, PCA, and SVDD, unlike the micro-classifiers, there is a one-to-one relationship between a cluster and an OCC. A minimum cluster size of 200 points is required in order to create a new ensemble member. This is to allow for a reasonable number of training points. Often in OCCs an error on the training data is allowed (typically 10%). This helps to prevent over-fitting. The allowed error on each classifier here is 0% as the training data has been 'pre-processed' by the act of clustering and its unlikely that noise/ outlier points are clustered.

In MST, the main tunable parameter N determines the number of paths of maximum length allowed in the tree. In each experiment, the entire MST is allowed to be

the maximum length which removes the need for a tunable parameter. Similarly, in PCA, the fraction of explained variance by the principal components is set to 90%, this removes any tunable parameters and allows for easier model creation. With the SVDD it is not so easy. A Radial Bias Function is used as the kernal and this requires a width parameter σ which is sensitive and can greatly affect the model. In order to find a good value for σ, grid-search with k-fold cross validation is used. This adds additional computation and complexity to the framework but without a suitable parameter the model can give a very bad performance.

Each homogeneous ensemble with different base classifiers is evaluated on the Network Intrusion Stream and the Forest Cover Stream. On the Network stream (Figure 6.9) the micro-classifier performs the best of the four, followed by MST, then PCA with SVDD performing the worst. However, on the Forest Cover Stream (Figure 6.10) SVDD performs the best, marginally better than the micro-classifier. MST performs poorly on this data-stream.

6.6 Summary

This chapter described a framework for classifying dynamic data streams with a scarcity of labels. Clustering and One-Class Classification Ensemble Learning (COCEL) consists of a stream-clustering algorithm and an ensemble of one-class classifiers with active learning. COCEL was tested on synthetic dynamic data streams exhibiting concept evolution, virtual concept drift, and real concept drift. The proposed framework performed much better than a static ensemble and achieved high classification accuracy despite requiring less than 0.01% of the stream's labels. It was further evaluated on two real data-streams which would be realistic applications of the COCEL framework; one high-velocity stream where manually labelling each incoming point is not feasible, and another where manually labelling the stream is expensive and labour intensive. Again, the proposed framework outperforms a static ensemble while requiring a tiny fraction of the stream to be labelled. Finally, the results of COCEL Were compared with the results of two state-of-the-art fully supervised ensembles (i.e. they require 100% of the data to be labelled). COCEL was shown to achieve comparative accuracy on three of the four data-streams.

In the experiments described, MDSC was used with a homogeneous ensemble of classifiers. A time-weighted lazy learner was introduced. This micro-classifier is an extension of the micro-clusters discovered by MDSC. Three further OCCs were evaluated; a Minimum Spanning Tree, a Support Vector Domain Description and Principal Component Analysis for OCC. Theoretically, any stream-clustering

algorithm and any ensemble of OCCs should work, though there are some practicalities that would need to be considered. For example, the "no-free-lunch" theorem certainly applies (as evidenced with SVDD's performance in both streams) but a heterogeneous ensemble could mitigate this. Or a suitable type of OCC could be pre-selected during the training phase, COCEL allows for any OCC to be used.

Some OCCs have been shown to work well with fewer training samples [95], others require a large training set [150]. Some OCCs require fine-tuned parameters and traditional tuning techniques (cross-validation, Leave-One-Out, etc) with large training sets might not be feasible in a high-velocity stream due to memory constraints, or for example, a model that takes a long time to train might be already redundant by the time it is added to the ensemble.

When labelling is expensive, time-consuming, labour-intensive, or simply not feasible, COCEL is potentially a worthwhile alternative to fully-supervised ensemble learning .

Chapter 7

Conclusion

Mining and analysing data in real-time is a necessary progression from traditional data mining. However, data stream mining requires additional considerations to traditional batch-mining. Ideally, the stream will be analysed in real time and patterns and anomalies can be observed as and when they happen. This imposes strict time constraints on a streaming algorithm and usually only a single pass of the data is afforded. Memory constraints also exist; a stream is potentially infinite but only a finite amount of memory is available. This constraint requires some form of data-summarisation and a 'forgetting' mechanism.

Along with time and memory constraints, change is a major consideration when mining data-streams. Data streams can be stationary but typically some form of change is expected and streams are usually assumed to be dynamic. The term 'dynamic' refers to the concepts or classes within the stream. For example, over time additional concepts can appear in the stream. These new concepts should be recognised as being new and the underlying model updated accordingly. This type of change is referred to concept evolution. Another type of change in a stream can occur in the form of concept drift whereby the characteristics of the data change or there is a change in the relationship between the data and the target class.

The challenge of recognising and reacting to change in a stream is compounded by the *scarcity of labels* problem. This refers to the very practical situation in which the underlying, ground-truth of every incoming instance in a stream cannot be known. For example, in a high-velocity stream it is unrealistic to assume that every incoming point can be manually labelled by a human expert. It might be easy to label one (or a small number of them) but not feasible to label them all. Similarly, in some cases the associated label of an incoming point cannot be known until a later time (for example, credit records and loan repayments), this is referred to as *verification latency*. In the extreme case the ground-truth may never be known

at all.

Much work has been done on supervised classification in dynamic data streams. In the supervised setting, it is assumed that the true class label of each incoming point is immediately available and easily accessible though there are few practical situations where this assumption holds. One scenario might be in financial prediction where the value of a stock at each time-step is readily available and can be used to update a classification model. However, the vast majority of streams will suffer from a scarcity-of-labels (remote sensing, cyber-security, web-usage, social-media, condition monitoring, climate analysis, etc.) .

The goal of this thesis was to evaluate unsupervised learning as the basis for online classification in dynamic data-streams with a scarcity of labels. To achieve this goal two dynamic stream-clustering algorithms were developed: Ant Colony Stream Clustering (ACSC) and Multi-Density Stream Clustering (MDSC). These algorithms can discover non-stationary clusters and cope with change at the concept level (concept drift and concept evolution). They are both density based and therefore suffer from the 'curse of dimensionality' whereby high-dimensional data renders distance-based measurement and any of form of 'density' difficult. To overcome this, a Dynamic Feature Mask (DFM) was developed to 'sit-on-top' of these clustering algorithms allowing them to deal with high-dimensional data and also to track change at the feature level. Finally, a novel framework called Clustering and One-Class Classification Ensemble Learning (COCEL) was developed to perform dynamic classification in data streams with a scarcity of labels.

7.1 Summary of Results

7.1.1 Ant Colony Stream Clustering

Chapter 3 presents an ant-inspired approach to clustering non-stationary data streams. The proposed algorithm (ACSC), is based on the concept of artificial ants which identify clusters as 'nests' of micro-clusters in dense areas of the data. The algorithm uses a sliding window model and consists of two steps: 1) rough clusters (nests) are identified in a single pass of the window using a stochastic sampling method, 2) nests are sorted using an extension of the classic pick-and-drop model inspired by the sorting behaviour of ants.

- ACSC is shown to outperform other ant-based clustering algorithms (ACA, ACAm, ATTA, AntClust) over three static datasets.
- ACSC outperforms other stream-clustering algorithms (DenStream, CluStream

and ClusTree) over six non-stationary data streams. The levels of cluster purity are comparable across each stream but ACSC achieves better F-Measure and Rand Index scores. The second phase of the algorithm, the sorting phase, is the reason for this. The probabilistic functions for picking and dropping micro-clusters are biased towards the dissolution of smaller clusters and incorporating them into similar, larger clusters. This improves the precision and recall scores (and hence the F-Measure) and creates clusters closer to the true structure of the data. This is reflected in the Rand Index score.

- ACSC was shown to process streams faster than the comparative algorithms. This is due to the stochastic sampling method used and also how micro-clusters attempt to merge. This merging operation is expensive. In ACSC, only micro-clusters in the same cluster attempt to merge, replacing an exhaustive search and reducing the number of failed merging operations.

- ACSC performs favourably to the comparative algorithms, requires fewer parameters and considerably fewer calculations ($\approx$ 10 times fewer calculations)

7.1.2 Multi-Density Stream Clustering

Chpater 4 describes Multi-Density Stream Clustering (MDSC). MDSC extends some of the swarm intelligence inspired aspects of density clustering introduced in ACSC (e.g. ants forming nests). MDSC improves ACSC in two ways: clusters are online and the ϵ parameter is adaptive and local to each cluster. The adaptive ϵ has two benefits: 1) it removes the need to tune a sensitive, data-dependent parameter, and 2) it allows the discovery of multi-density clusters. Furthermore, because MDSC is online, identified clusters can be labelled and their dynamic behavior can be tracked.

- MDSC was shown to perform favourably to ACSC and two further peer algorithms: CEDAS and MuDi. If a stream contains a single density, the performance is comparable with ACSC. However, MDSC finds a better clustering solution on streams which contain multi-density clusters. The reason for this is the adaptive ϵ and the way it is identified. Clusters are discovered in order of density, i.e., more compact clusters are identified first. This allows the discovery of clusters (with a smaller ϵ) embedded in larger sparser clusters (with a higher ϵ).

- MDSC was qualitatively evaluated on a real-life air-quality monitoring stream. Results show that it can identify and track underlying patterns despite seasonal changes and cyclic behaviour. MDSC could discover and track patterns such

as rush-hour and a cyclic increase in pollutants during winter. A correlative relationship between certain pollutants and an inverse relationship between others were observed.

7.1.3 Dynamic Feature Mask

Chapter 5 presents a Dynamic Feature Mask (DFM) for unsupervised dynamic feature selection in non-stationary data-streams. Redundant features are masked and clustering is performed along unmasked, relevant features. If a feature's perceived importance changes, the mask is updated accordingly; previously unimportant features can be unmasked and features which loose relevance can become masked. The method is proposed to address two challenges in data-stream clustering: 1) feature drift - a change at the feature level in a stream, and 2) the problem of clustering high-dimensional streams where the curse of dimensionality renders distance measurements and the concepts of 'density' difficult.

- DFM was evaluated on three density based clustering algorithms: ACSC, MDSC, and CEDAS across four high-dimensional streams; two text streams and two image streams. In each case, the proposed DFM improves clustering performance and reduces the processing time required by the underlying algorithm.

- An unsupervised feature selection (FS) method is required to create and maintain the DFM, three static FS methods were evaluated: Laplacian Score, Multi-Cluster Feature Selection, and Maximum Variance. Experimental results suggest that on the lower dimensional ($\approx 1,000$ dimensions) streams, MCFS is the best selector for the mask. On the higher dimensional text streams (up to 60,000 dimensions), the Maximum Variance method selects the best features to maintain the mask. The Laplacian Score did not return the best features on any stream and was shown to require considerably more time than the other two methods.

- On each stream, the DFM was compared with a static feature mask. In the static case, the mask is created on one window at the beginning of the stream and is never updated. The dynamic mask performs better on each stream. On the higher dimensional streams, the static mask is preferable to no mask (without a mask the clustering algorithms could not return a solution at all) but on the lower dimensional streams it is preferable to use no mask rather than a static mask.

7.1.4 Clustering and One-Class Classification Ensemble Learning

Chapter 6 outlines a framework for classifying dynamic data streams with a scarcity of labels. Clustering and One-Class Classification Ensemble Learning (COCEL) consists of a stream-clustering algorithm and an ensemble of one-class classifiers with active learning.

- COCEL was tested on synthetic dynamic data streams exhibiting concept evolution, virtual concept drift, and real concept drift. The proposed framework outperforms a static ensemble (an ensemble created with a training set but never updated) while requiring a tiny fraction of the stream to be labelled.
- COCEL was compared with two state-of-the-art fully supervised ensembles (i.e., they require 100% of the data to be labelled): Adaptive Random Forest and OzaBoost. COCEL was shown to achieve comparative accuracy on three of the four data-streams despite requiring at most 0.004% of the stream's labels.
- The framwowrk was further compared with a peer active-learning algorithm for data streams; The Active Classifier. With the same labelling budget, COCEL achieved a much better performance.
- Four base OCCS were evaluated for use in a homogeneous ensemble; Support Vector Domain Description, Principal Component Analysis, Minimum Spanning Trees, and a Time-Weighted Micro-Classifier. Overall, the Micro-Classifier performed the best but results on different streams show a high variance in their performance suggesting certain models are better suited to certain streams.

7.2 Unique Contribution

The work in this thesis contributes to three separate sub-fields in dynamic data-stream analysis; clustering, feature selection and classification.

7.2.1 Dynamic Stream Clustering

ACSC and MDSC are proposed as novel stream clustering algorithms. These algorithms aim to address the current imbalance in the literature between ant-based

static clustering algorithms (of which there are many) and ant-based stream clustering algorithms (of which there is barely any).

Both algorithms are density based. The majority of density stream clustering methods adopt the two phase, on-line/off-line approach whereby data is first summarised online and then clustered offline. The disadvantages of this approach are that a) the behaviour of clusters cannot be tracked over time and b) only clusters of a single density can be discovered. Recent proposals to discover multi-density clusters in streaming data require sensitive, static parameters and use the two-phase approach. The two-phase approach has been combined into a single online phase, potentially (with algorithmic extensions) allowing for on-line tracking, however they are unable to deal with multi-density clusters. MDSC proposed in this thesis aims to fill the space here; an online method allowing multi-density clusters to be discovered and tracked over time.

7.2.2 Dynamic Feature Selection

The majority of research on dynamic FS for data-streams assume the supervised method [96, 144, 33, 137] and is typically used for classification tasks and not suitable for clustering. Existing stream-clustering algorithms can deal with change at the concept level (concept drift and concept evolution) but are not designed to track change at the feature level. Furthermore, they suffer from the curse of dimensionality and cannot cluster high-dimensional streams, *e.g.*, text or image streams.

The method proposed in this thesis aims to address these two challenges: tracking change at the feature level and dynamically clustering in high dimensions.

7.2.3 Dynamic Classification with a Scarcity of Labels

A novel framework for classifying dynamic streams with a scarcity of labels is described. Other cluster/classification hybrid ensembles have been proposed in the literature but the unique contribution of this work is to use a stream-clustering algorithm to detect and react to change (as opposed to sliding windows and k-means in previous proposals).

An adaptive time-weighted micro-classifier based on the clusters discovered by MDSC is described as a novel One-Class Classifier.

7.3 Limitations

Multi-Density Stream Clustering described in Chapter 4 was evaluated with synthetic data, popular benchmark data-sets and also a real-world stream. None of the other chapters were evaluated in case studies using real-world applications. Although the presented results are convincing, each component (Ant Colony Stream Clustering, the Dynmaic Feature Mask and the COCEL framework) would be more credible if the results on benchmark data were replicated on real-world applications. This would be particularly true if the proposed methods were implemented in a case-study which highlighted why traditional methods were not suitable but their streaming counterparts were.

A second limitation is with the COCEL framework proposed in Chapter 6. COCEL was shown to detect and react to concept evolution and also virtual concept drift (a change in $P(x)$ leading to a change in $P(y|x)$) however if the stream contains sudden real concept drift (a change in $P(y|x)$ but no change in $P(x)$) then the framework would be unable to recognise this change.

7.4 Wider Impact

The prevalence of mobile phones, social media, remote sensors, financial transactions, climate monitoring, etc., has resulted in an enormous amount of data being generated in real time and typically this data is generated in a streaming fashion. The transition from classical data-analysis to real-time analysis is necessary to accommodate an increase in data. This increase in data presents great opportunity for researchers and practitioners to improve their decision making, model building, and monitoring capabilities. However some fundamental challenges need to be addressed before this potential can be fully realised.

This thesis contributes to the advancement in this direction and has application in many important fields. For example, the increasing potential for data analysis in medical applications like patient monitoring [10] and clinical data analysis [15]. In these areas it is easy to imagine a difficulty with label gathering and certainly the label verification latency problem exists, and the reaction of patients to treatment is non-stationary. In the field of astronomy data is increasingly being generated in huge streams, for example in 2022 the Large Synoptic Survey Telescope will begin operation in Chile and will stream data at a rate of two terabytes per hour [85]. Analysing and summarising this data in real time is a more attractive alternative to storing it all and analysing in batches. A further application area where this research can impact is the growing prevalence of the Internet Of Things. Data generators like

sensor networks, vehicle telematics, energy prediction, and environmental remote sensing are all subject to label latency and an intrinsic non-stationary; for example the task of predicting energy demand is non-stationary and is affected by climate fluctuations, increasing populations, improvements to grid efficiency and disruptive technologies like solar powered homes returning energy to the grid.

7.5 Future Work

There are a number of avenues for future research leading from the ideas developed in this thesis, particularly within the clustering-and-classification ensemble model. Currently, COCEL actively requests a label only when a new cluster is discovered. This keeps labelling costs very low (especially in times of stability). However, a higher labelling budget leads to a higher classification performance. This is intuitive and has been empirically shown ([209]) but COCEL does not allow for additional labels (if they are available) to be used by the model. An interesting extension would be to introduce a labelling budget to the framework allowing the ensemble to periodically check the validity of its predictions. The ensemble could be updated and improved based on these validation checks. This measure could potentially solve the problem of sudden, real concept drift described in Section 7.3 (Limitations).

A further extension to COCEL would be to investigate a heterogeneous ensemble of one-class classifiers. The experiments showed a variance in the performance of classifiers on each stream and this could be attributed to the "no free lunch" theory. A heterogenous ensemble could mitigate this, for example micro-classifiers require very few training samples and could be created and deployed as soon as change is detected while more complex classifiers (auto-encoders, support vector machines, etc.) could be created when sufficient training samples have been collected. This could allow for stable concepts to be modelled effectively (with a large number of training samples) with smaller, more lightweight models reacting quickly to change. The type of classifier created could be automated and determined by a fuzzy inference system based on stream velocity, stream volatility, rate of change, number of members currently in the ensemble, etc.

Some of the more recent ideas in the field of machine learning can be incorporated into the work presented in this thesis. Zero-shot learning (ZSL) is a particularly relevant direction. ZSL is learning to recognise new concepts by just having a 'description' of them, this description could be a numeric vector or a set of linguistic tags. For example, a model, having never been trained on images of a zebra (but having been trained to recognise a horse) could recognise a zebra by the description

'similar to a horse' and 'has black-and-white stripes'. This would be very applicable in streams which contain concept evolution or in streams with substantial concept drift. Another modern approach to the Neural Network is the Long-Short Term Memory Neural Network (LSTM), this approach can not only process single data points(as in traditional Neural Networks) but can also process sequences of data. 'Memory' is incorporated into the model along with a 'forgetting' mechanism making it useful in temporal, dynamic data streams. LSTM along with an adequate change detection method (for example a companion stream-clustering algorithm) could be used as a powerful method to process complex data streams.

Another interesting research direction could be the analysis of multiple related streams simultaneously. For example, an air-quality stream, a traffic monitoring stream, and live weather data could be analysed in parallel with a meta learner discovering underlying rules and relationships. Another example could be in financial prediction with streams of data from the stock market, financial news reports and social media mentions of a particular stock. Analysing these streams in parallel could discover relationships and create better predictors.

Bibliography

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams", Proc. 29th Int. Conf. Very Large Data Bases, vol. 29, pp. 81–92, 2003.

[2] C. Aggarwal, J. Han, J. Wang, and P.S. Yu." A framework for projected clustering of high dimensional data streams. I Proceedings of the Thirtieth international conference on Very large data bases - Volume 30. VLDB Endowment, Toronto, Canada, 852–863. 2004.

[3] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu. "On demand classification of data streams" In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 503-508). ACM. 2004

[4] S. Alelyani, J. Tang, and H. Liu, "Feature Selection for Clustering: A Review", Data Clustering: Algorithms and Applications, 29, pp. 110-121, 2013.

[5] C. Alippi and M. Roveri, "An adaptive cusum-based test for signal change detection" in Proc. Int. Symp. Circuits Systems,pp. 1–4. 2006.

[6] C. Alippi, "Intelligence for Embedded Systems". Berlin, Germany: Springer-Verlag, 2014.

[7] C. Alippi, G. Boracchi, and M. Roveri, "Just-in-time classifiers for recurrent concepts" IEEE Trans. Neural Netw. Learn. Syst., vol. 24, no. 4, pp. 620–634, 2013.

[8] T. Al-Khateeb, M.M. Masud, L. Khan, C. Aggarwal, C., J.Han, and B. Thuraisingham. " Stream classification with recurring and novel class detection using class-based ensemble" In 2012 IEEE 12th International Conference on Data Mining (pp. 31-40). IEEE. 2012.

[9] T. Al-Khateeb, M.M. Masud, L. Khan,and B. Thuraisingham. "Cloud guided stream classification using class-based ensemble". In Cloud Computing

(CLOUD), 2012 IEEE 5th International Conference on (pp. 694-701). IEEE. 2012

[10] M.C. Almodovar "Streaming Analytics in Pediatric Cardiac Care". In Critical Heart Disease in Infants and Children (pp. 18-23). Elsevier. 2019.

[11] Amazon Mechanical Turk `https://www.mturk.com/`, 03 12 2018.

[12] A. Amini, et al. "MuDi-Stream: A multi density clustering algorithm for evolving data stream," Journal of Network and Computer Applications, vol. 59, pp. 370–385, 2016.

[13] M. Ankerst, M.M Breunig, H.P Kriegel and J. Sander. "OPTICS: ordering points to identify the clustering structure" In ACM Sigmod record (Vol. 28, No. 2, pp. 49-60). 1999.

[14] P. Armitage and P. Armitage, "Sequential Medical Trials". Oxford, U.K.: Blackwell, 1975

[15] S.Bahri, N. Zoghlami, M. Abed, and J.M.R. Tavares. "BIG DATA for Healthcare: A Survey". IEEE Access, 7, pp.7397-7408. 2019

[16] T. Ban and S. Abe. "Implementing multi-class classifiers by one-class classification methods", InInternational Joint Conference on Neural Networks, pages 327–332, 2006.

[17] M. Belkin and P. Niyogi, P., "Laplacian eigenmaps and spectral techniques for embedding and clustering". In Advances in neural information processing systems, pp. 585-591, 2002.

[18] J.M. Bernardo and A.F. Smith, Bayesian theory 2001.

[19] J. Bertini Jr, A. Lopes and L. Zhao, "Partially labeled data stream classification with the semi-supervised K-associated graph", Springer-Journal of Brazilian Computer Society, 2012. Information and Knowledge Management, pp. 1031-1040, October 2015.

[20] J. Biesiada and W. Duch,"Feature selection for high-dimensional data; a Pearson redundancy based filter",In Computer Recognition Systems 2, Springer, Berlin, Heidelberg, pp. 242-249, 2007.

[21] A. Bifet and R. Gavalda, "Kalman filters and adaptive windows for learning in data streams," in Proc. Int. Conf. Discovery Science, pp. 29–40. 2006

[22] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing" in Proc. SIAM Int. Conf. Data Mining, 2007.

[23] A. Bifet, G. Holmes, B. Pfahringer and R. Gavalda. "Improving adaptive bagging methods for evolving data streams". In Asian conference on machine learning (pp. 23-37). Springer, Berlin, Heidelberg.2009.

[24] A. Bifet, G. Holmes, R. Kirkby, R. and B. Pfahringer,"Moa: Massive online analysis." Journal of Machine Learning Research, pp.1601-1604, 2010

[25] C. Bishop.Neural Networks for Pattern Recognition. Oxford Uni-versity Press, Walton Street, Oxford OX2 6DP. 1995

[26] U. Boryczka, "Finding groups in data: Cluster analysis with ants" Applied Soft Computing, vol. 9, no. 1, pp. 61–70, Jan. 2009.

[27] M.-R. Bouguelia, Y. Belaïd, and A. Belaïd "An adaptive incremental clustering method based on the growing neural gas algorithm". ICPRAM, pp. 42-49. 2013

[28] D. Brzezinski and J. Stefanowski, "Reacting to different types of concept drift: the accuracy updated ensemble algorithm". IEEE Trans. Neural Netw. Learn. Syst. 25 (1) 81–94, 2014.

[29] D. Cai, C. Zhang, and X. He, "Unsupervised feature selection for multi-cluster data", In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 333-342, July, 2010.

[30] F. Camci, and R.B. Chinnam, "General support vector representation machine for one-class classification of non-stationary classes" Pattern Recognition, 41(10), pp.3021-3034. 2008.

[31] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," SDM, vol. 6, pp. 328–339, 2006.

[32] C. Carmelo, et al. "Enhancing density-based clustering: Parameter reduction and outlier detection" Information Systems, vol. 38, no. 3, pp. 317–330, 2013.

[33] V.R. Carvalho and W.W. Cohen, "Single-pass online learning: Performance, voting schemes and online feature selection", In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 548-553, 2009.

[34] P. Casale, O. Pujol, and P. Radeva, P. "Approximate convex hulls family for one-class classification" In International Workshop on Multiple Classifier Systems (pp. 106-115). Springer, Berlin, Heidelberg, 2011.

[35] G. Cauwenberghs, T. Poggio. "Incremental and decremental support vectormachine learning", in: Proc. NIPS, 2001.

[36] Y. Chen, & L. Tu, "Density-based clustering for real-time stream data." In Proceedings of the 13th ACM SIGKDD internationalACM. August, 2007

[37] V. Cheplygina, DMJ Tax. "Pruned random subspace method for one-class classifiers". In: Sansone C, Kittler J, Roli F (eds) Multiple classifier systems. Lecture notes in computer science, vol 6713. Springer, Berlin, pp 96–10, 2011.

[38] S.C. Chu, J.F. Roddick, C.J. Su, and J.S. Pan" Constrained ant colony optimization for data clustering" In Lecture notes in artificial intelligence: 8th pacific rim international conference on artificial intelligence(pp. 534–543).2004.

[39] L. Cohen, G. Avrahami-Bakish, M. Last, A. Kandel, and O. Kipersztok, "Real-time data mining of non-stationary data streams from sensor networks" Inform. Fusion, vol. 9, no. 3, pp. 344–353,2008.

[40] E. Cohen and M. Strauss, "Maintaining time-decaying stream aggregates" in Proc. 22nd ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Systems,pp. 223–233. 2003.

[41] B. Cyganek. "Image segmentation with a hybrid ensemble of one-class support vector machines" In: Proceedings of 5th international conference on Hybrid artificial intelligence systems (HAIS'10), part I, San Sebastián, pp 254–261. 2010

[42] X. Dang, V.Lee, W. Ng, A. Ciptadi and K. Ong. "An em-based algorithm for clustering data streams in sliding windows" In Database Systems for Advanced Applications. Lecture Notes in Computer Science, vol. 5463. Springer Berlin / Heidelberg, 230–235. 2009.

[43] T.G. Dietterich. "Ensemble methods in machine learning". In International workshop on multiple classifier systems, (pp. 1-15). Springer, Berlin, Heidelberg. 2000.

[44] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. "Learning in nonstationary environments: A survey" IEEE Computational Intelligence Magazine, 10(4), pp.12-25. 2015.

[45] A.P. Dempster, N.M. Laird, and D.B. Rubin,"Maximum Likelihood from Incomplete Data via the EM Algorithm". Journal of the Royal Statistical Society, Series B. 39 (1): 1–38. 1977

[46] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien, "The dynamics of collective sorting robot-like ants and ant-like robots" Proceedings of the 1st International Conference on Simulation of Adaptive Behavior From Animals to Animats, pp. 356–363, 1991.

[47] S. Ding et al. "An adaptive density data stream clustering algorithm." Cognitive Computation, vol. 8, no. 1, pp. 30–38, 2016.

[48] M. Dorigo, M. Birattari, and T. Stizle,"Ant colony optimization" IEEE Comput. Intell. Mag., vol. 1, no. 4, pp. 28–39, 2006.

[49] P. Domingos and G. Hulton, "Mining high-speed data stream" in Proc. 6th ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining,pp. 71–80. 2000.

[50] R. Duda and P. Hart, "Pattern Classification and Scene Analysis" John Wiley & Sons, New York. 1973

[51] M. Ester, H. P. Kriegel, J. Sander, and X. Xu,"A density-based algorithm for discovering clusters in large spatial databases with noise" KDD, vol. 96, pp. 226–231, 1996.

[52] G. Esfandani, H. Abolhassani. "MSDBSCAN: multi-density scale-independent clustering algorithm based on DBSCAN," Proc. Int. Conf. on Advanced Data Mining and Applications, 2010.

[53] R. Elwell , R. Polikar , Incremental learning of concept drift in nonstationary environments, IEEE Trans. Neural Netw. 22 (10). pp 1517–1531. 2011.

[54] A. H. Fahim, A. M. Salem, F. A. Torkey, and M. A. Ramadan, "Density Clustering Based on Radius of Data (DCBRD)" World Academy of Science, Engineering and Technology, 2006.

[55] C. Fahy and S. Yang. "Dynamic Stream Clustering Using Ants" In Advances in Computational Intelligence Systems (pp. 495-508). Springer, Cham. 2017

[56] C. Fahy, S. Yang, and M. Gongora. Ant colony stream clustering: A fast density clustering algorithm for dynamic data streams. IEEE Transactions on Cybernetics, 49(6): 2215-2228, June 2019 (DOI: 10.1109/TCYB.2018.2822552)

[57] C. Fahy, S. Yang, and M. Gongora. "Finding multi-density clusters in non-stationary data streams using an ant colony with adaptive parameters." In Proc. 2017 IEEE Congress on Evolutionary Computation, pp. 673-680, 2017.

[58] C. Fahy and S. Yang. "Finding and Tracking Multi-Density Clusters in Data Streams" IEEE Trans. on Big Data, accepted, May, 2019.

[59] C. Fahy and S. Yang. "Dynamic Feature-Selection for Clustering High Dimensional Data-Streams" IEEE ACCESS, submitted June, 2019.

[60] C.Fahy and S. Yang. "Classification in Dynamic Data Streams with a Scarcity of Labels" IEEE Trans. on Pattern Matching and Machine Intelligence, submitted April, 2019.

[61] W. Fan, Y.A Huang, H. Wang, and P.S. Yu."Active mining of data streams". In Proceedings of the 2004 SIAM International Conference on Data Mining (pp. 457-461). Society for Industrial and Applied Mathematics. 2004.

[62] D. Fernández-Francos, O. Fontenla-Romero, and A. lonso-Betanzos,"One-class convex hull-based algorithm for classification in distributed environments" IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2017

[63] Figure-Eight `https://www.figure-eight.com/`, 03 12 2018.

[64] J. Fiscus, G. Doddington, J. Garofolo and A. Martin, "NIST's 1998 Topic Detection and Tracking evaluation (TDT2)", In Proceedings of the 1999 DARPA Broadcast News Workshop pp. 19-24, 1999.

[65] S. Furao, T. Ogura, and O. Hasegawa. "An enhanced self-organizing incremental neural network for online unsupervised learning". Neural Networks, 20(8), pp.893-903. 2007

[66] A. Forestiero, C. Pizzuti, and G. Spezzano, "A single pass algorithm for clustering evolving data streams based on swarm intelligence," Data Mining and Knowledge Discovery, vol. 26, no. 1, pp. 1–26, Nov. 2011.

[67] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection" in Proc. Advances Artificial Intelligence–SBIA, pp. 286–295. 2004.

[68] M. Galar, A.Fernández, E. Barrenechea,S. Bustince, F. Herrera. "Dynamic classifier selection for one-vs-one strategy: avoiding non-competent classifiers". Pattern Recognit 46(12):3412–3424 2014.

[69] C. Gautam and A. Tiwari, "On the construction of extreme learning machine for one class classifier". In Proceedings of ELM-2015 Volume 1 (pp. 447-461). Springer, Cham. 2016.

[70] J. Gehrke, R. Ramakrishnan, and V. Ganti.RainForest - a framework for fast decision treeconstruction of large datasets. InVLDB '98, pages416–427, 1998.

[71] M. Ghesmoune, M. Lebbah, and H. Azzag. "A new growing neural gas for clustering data streams". Neural Networks, 78, pp.36-50. 2016.

[72] H.M.,Gomes, A. Bifet, J Read, J.P Barddal,F. Enembreck, B. Pfharinger, G. Holmes and T. Abdessalem. "Adaptive random forests for evolving data stream classification" Machine Learning, 106(9-10), pp.1469-1495. 2017

[73] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures" Neural Netw., vol. 1, no. 1, pp. 17–61, 1988.

[74] Q. Gu, Z. Li, and J. Han, "Generalized Fisher score for feature selection", In: Proceedings of the 27th International Conference on Uncertainty in Artificial Intelligence (UAI'11), pp. 266-273, 2011.

[75] S. Guha, R. Rastogi and K. Shim. "CURE: an efficient clustering algorithm for large databases". In ACM Sigmod Record (Vol. 27, No. 2, pp. 73-84). 1998.

[76] M. Hall, "Correlation-based feature selection for machine learning", PhD Thesis, New Zealand Department of Computer Science, Waikato University, 1999.

[77] J. Handl, J. Knowles, and M. Dorigo, "Ant-based clustering and topographic mapping" Artif. Life, vol. 12, no. 1, pp. 35–62, Jan. 2006.

[78] J. Handl and B. Meyer, "Ant-based and swarm-based clustering" Swarm Intell., vol. 1, no. 2, pp. 95–113, Nov. 2007.

[79] M. Harel, K. Crammer, R. E l-Ya n iv, and S. Mannor, "Concept drift detection through resampling" in Proc. Int. 31st Conf. Machine Learning, p. 1009–1017, 2014

[80] J.A. Hartigan and M.A. Wong. "Algorithm AS 136: A k-means clustering algorithm" Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1), pp.100-108. 1979

[81] D. M. Hawkins, Q. Peihua, and W. K . Chang, "The changepoint model for statistical process control" J. Qual. Technol., vol. 35, no. 4, pp. 355–366, Oct. 2003.

[82] M.J.Hosseini, A. Gholipour and H. Beigy, "An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams" Springer-Knowledge Info. Systems, 2015.

[83] C. L. Huang,and C.J. Wang, "A GA-based feature selection and parameters optimization for support vector machines", Expert Systems with applications, vol. 31, no. 2, pp. 231-240, 2006.

[84] H. Huang, S. Yoo, and S.P. Kasiviswanathan, "Unsupervised feature selection on data streams", In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 1031-1040, 2015.

[85] P. Huijse, P.A, Estevez, P. Protopapas, J.C. Principe, and P. Zegers. "Computational intelligence challenges and applications on large-scale astronomical time series databases". IEEE Computational Intelligence Magazine, 9(3), pp.27-39. 2014

[86] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data stream" in Proc. Conf. Knowledge Discovery Data, pp. 97–106. 2001.

[87] R. Hyde, P. Angelov, and A. R. MacKenzie, "Fully online clustering of evolving data streams into arbitrarily shaped clusters," Information Sciences, vol. 382-383, pp. 96–114, 2017.

[88] D. Ienco, A. Bifet, I. Žliobaitė and B. Pfahringer. "Clustering based active learning for evolving data streams". In International Conference on Discovery Science (pp. 79-93). Springer, Berlin, Heidelberg. 2013.

[89] W. B. Innes, "Effect of nitrogen oxide emissions on ozone levels in metropolitan regions." Environmental science and technology 15.8, pp. 904-912. 1981.

[90] J-Profiler: Java Profiler. `https://www.ej-technologies.com/products/jprofiler/overview.html`, 21 11 2017.

[91] N. Jardine and C. J. van Rijsbergen, "The use of hierarchic clustering in information retrieval," Information Storage and Retrieval, vol. 7, no. 5, pp. 217–240, Dec. 1971.

[92] G.M. Jenkins and D.G. Watts. Spectral analysis. 1968.

[93] H. Jiang, J. Li, S. Yi, X. Wang, and X. Hu, "A new hybrid method based on partitioning-based DBSCAN and ant clustering" Expert Syst. with Appl., vol. 38, no. 8, pp. 9373–9381, Aug. 2011.

[94] B. Junior, and M. do Carmo Nicoletti. "An iterative boosting-based ensemble for streaming data classification". Information Fusion, 45, pp.66-78, 2019

[95] P. Juszczak, D.M.Tax,E. Pe, and R.P. Duin, "Minimum spanning tree based one-class classifier" Neurocomputing, 72(7-9), pp.1859-1869. 2009.

[96] I. Katakis, G. Tsoumakas, and I. Vlahavas, "On the utility of incremental feature selection for the classification of textual data streams", In Panhellenic Conference on Informatics, pp. 338-348, November 2005.

[97] L. Kaufman and P.J. Rousseeuw. "Clustering by means of Medoids" in Statistical Data Analysis Based on the L1, L1–Norm and Related Methods" pp 405–416. 1989.

[98] L. Kaufman and P.J. Rousseeuw. "Partitioning around medoids. Finding groups in data: an introduction to cluster analysis" pp.68-125. 1990.

[99] G. Karypis, E.H. Han and V. Kumar. "Chameleon: Hierarchical clustering using dynamic modelling". Computer, 32(8), pp.68-75, 1999.

[100] S.S. Khan and M.G. Madden, "One-class classification: taxonomy of study and review of techniques". The Knowledge Engineering Review, 29(3), pp.345-374. 2014.

[101] R. Klinkenberg and T. Joachims. "Detecting Concept Drift with Support Vector Machines" In ICML (pp. 487-494). 2000.

[102] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weight-ing" Intell. Data Anal., vol. 8, no. 3, pp. 281–300, 2004.

[103] T. Kohonen. "Self-organized formation of topologically correct feature maps" Biological Cybernetics, 43 (1), pp. 59-69. 1982

[104] J.Z. Kolter and M.A. Maloof. "Using additive expert ensembles to cope with concept drift" in: Proceedings of the Twenty Second ACM International Conference on Machine Learning (ICML'05),pp. 449–456 . Bonn, Germany. 2005.

[105] J.Z. Kolter and M.A. Maloof. "Dynamic weighted majority: an ensemble method for drifting concepts" J. Mach. Learn. Res. 8.pp 2755–2790. 2007.

[106] M. Korürek and A. Nizam, "A new arrhythmia clustering technique based on ant colony optimization" J. of Biomedical Inform., vol. 41, no. 6, pp. 874–881, Dec. 2008.

[107] S.B. Kotsiantis, I. Zaharakis,and P. Pintelas. “Supervised machine learning: A review of classification techniques”. Emerging artificial intelligence applications in computer engineering, 160, pp.3-24.1996.

[108] I. Koychev, “Gradual forgetting for adaptation to concept drift” inProc. ECAI Work-shop Current Issues Spatio-Temporal Reasoning,pp. 101–106. 2000.

[109] B. Krawczyk. “One-class classifier ensemble pruning and weighting with firefly algorithm”. Neurocomputing 150:490–500. 2015.

[110] B. Krawczyk, M. Woźniak, B. Cyganek “Clustering-based ensembles for one-class classification”. Inf Sci 264:182–195. 2015.

[111] B. Krawczyk and B. Cyganek “Selecting locally specialised classifiers for one-class classification ensembles” Pattern analysis and applications, 20(2), pp.427-439. 2017.

[112] P. Kranen, I. Assent, C. Baldauf and T. Seidl “The ClusTree: indexing micro-clusters for anytime stream mining“ Knowledge and information systems, 29(2), pp.249-272. 2011

[113] I. Kuncheva, “Classifier ensembles for changing environments” in: Proceedings of the 5th MCS International Workshop on Multiple ClassifierSystems, in: Lecture Notes in Computer Science, 3077, Springer.pp. 1–15 . 2004.

[114] N. Labroche, N. Monmarche, and G. Venturini, “AntClust: ant clustering and web usage mining” Pro. 2003 Genetic and Evol. Comput. Conf., pp. 25–36, 2003.

[115] M. Last. “Online classification of non-stationary data streams”. Intelligent data analysis, 6(2), pp.129-147. 2002.

[116] M.H. Law, M.A Figueiredo, and A.K Jain, “Simultaneous feature selection and clustering using mixture models”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 9, pp. 1154-1166. 2004.

[117] C. Lee and G. G. Lee, “Information gain and divergence-based feature selection for machine learning-based text categorization”, Information processing & management, vol. 42, no. 1, pp. 155-165, 2006.

[118] Q. Leng, H. Qi, J. Miao, W. Zhu, and G. Su. “One-class classification with extreme learning machine” Mathematical problems in engineering, 2015.

[119] Q. Li, Z. Shi, J. Shi and Z. Shi. " Swarm intelligence clustering algorithm based on attractor" In Lecture notes in computer science: Advances in natural computation, first international conference. (pp. 496–504). 2005

[120] J. Li et al., "Feature selection: A data perspective", ACM Computing Surveys, vol. 50, no. 6, pp.94, 2017.

[121] P. Li, X. Wu and X. Hu, "Mining recurring concept drifts with limited labeled streaming data", In Proceeding of the 2ndAsian Conference on Machine Learning (ACML-JMLR), Tokyo, Japan, pp. 241–252. 2010.

[122] X. Wu, P. Lia and X. Hu, "Learning from concept drifting data streams with unlabeled data" Elsevier- Neurocomputing 92, p. 145-15. 2012

[123] N.Y. Liang, G.B. Huang, P. Saratchandran, N. Sundararajan." A fast andaccurate online sequential learning algorithm for feedforward networks" NN17 (6),1411–1423. 2006.

[124] L. Liu K. Jing K, Y. Guo "A three-step clustering algorithm over an evolving data stream" In Proc. the IEEE Int. Conf.Intelligent Computing and Intelligent Systems,pp.160-164. 2009.

[125] J. Liu, G. Xu, D. Xiao, L. Gu and X.X. Niu, "A semi-supervised ensemble approach for mining data stream" Journal Of Computers, vol. 8, no. 11,p. 2873-2879. 2013.

[126] J. Lin and H. Lin. "A density-based clustering over evolving heterogeneous data stream". In Proc. the 2nd Int. Colloquium on Computing, Communication, Control, and Management, pp.275-277. 2009.

[127] P. Lindstrom, S. J. Delany, and B. M. Namee, "Handling concept driftin a text data stream constrained by high labelling cost", in Proc. 23rd Florida Artif. Intell. Res. Soc. Conf., pp. 1–7. 2010.

[128] P. Lindstrom, B. Mac Namee and Delany. "Drift detection using uncertainty distribution divergence". Evolving Systems, 4(1), pp.13-25. 2013.

[129] N. Littlestone and M.K. Warmuth. "The weighted majority algorithm", Inf. Comput. 108 212–261 . 1994.

[130] B. Liu, "A Fast Density-Based Clustering Algorithm For Large Databases" Proc. 5th Int. Conf. Machine Learning and Cybern., 2006.

[131] V. Losing, B. Hammer and H. Wersing. "Incremental on-line learning: A review and comparison of state of the art algorithms" Neurocomputing, 275, pp.1261-1274. 2018.

[132] E. Lumar and B. Faieta, "Diversity and adaptation in populations of clustering ants" Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior: From Animals to Animats, vol. 3, pp. 489–508, 1994.

[133] R. Łysiak, M. Kurzyński, T. Wołoszynski. "Optimal selection of ensemble classifiers using measures of competence and diversity of base classifiers". Neurocomputing 126:29–35 2014

[134] S. Mahran and K. Mahar, "Using grid for accelerating density based clustering" Proc. 2008 IEEE Int. Conf. Computer and Inform. Tech., 2008.

[135] T.M Martinetz, S.G. Berkovich, K.J. Schulten, "Neural-gas network for vector quantization and its application to time-series prediction", IEEE Transactions on Neural Networks, 4 (4), pp. 558-569, 1993.

[136] N. Masmoudi, H. Azzag, M. Lebbah, B. Cyrille, and B. J. Maher, "How to use ants for data stream clustering," Proc. 2015 IEEE Cong. Evol. Comput., pp. 656–663, 2015.

[137] M. Masud, J. Gao, L. Khan, J. Han and B.M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints", IEEE Transactions on Knowledge and Data Engineering, vol. 23, no. 6, pp.859-874, 2011.

[138] M.M. Masud, J. Gao, L. Khan, J. Han and B. Thuraisingham, "A practical approach to classify evolving data streams: training with limited amount of labeled dat" In Proceeding of the 8thIEEE International Conference on Data Mining,pp. 929–934. 2008

[139] M.M. Masud M.M, "Facing the reality of data stream classification: coping with scarcity of labeled data", Springer-Knowledge Information Systems, 33(1):213–244. 2012

[140] L. L. Minku, A . P. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift". IEEE Trans. Knowledge Data Eng., vol. 22, no. 5, pp. 731–742. 2010.

[141] L. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift" IEEE Trans. Knowledge Discovery Data Eng., vol. 24, no. 4, pp. 619–633. 2012.

[142] R.T. Ng and J. Han. "CLARANS: A method for clustering objects for spatial data mining" IEEE transactions on knowledge and data engineering, 14(5), pp.1003-1016.2002.

[143] H. Nguyen, W. Ng, Y. Woon and H. Tran, "Concurrent semi-supervised learning of data streams" In Data Warehousing and Knowledge Discovery, Springer Berlin Heidelberg, p.445–459. 2011

[144] H.L Nguyen, Y.K. Woon, W.K. Ng and L. Wan, "Heterogeneous ensemble for feature drifts in data streams", In Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Berlin, Heidelberg. pp. 1-12, May 2012.

[145] H.L Nguyen, Y.K. Woon, and W.K. Ng, "A survey on data stream clustering and classification" Knowledge and information systems, 45(3), pp.535-569. 2015

[146] M. Nicolau and J. McDermott, "A hybrid autoencoder and density estimation model for anomaly detection" In International Conference on Parallel Problem Solving from Nature (pp. 717-726). Springer, Cham. 2016

[147] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, S. and R. Motwani, "Streaming-data algorithms for high-quality clustering". In Data Engineering, 2002. Proceedings. 18th International IEEE Conference on (pp. 685-694). 2002.

[148] R. Odate, H. Shinjo, Y. Suzuki and M. Motobayashi. "Semi-supervised Clustering Framework Based on Active Learning for Real Data". In Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR) (pp. 184-193). Springer, Cham. 2018.

[149] N. Oza and S. Russell. "Online bagging and boosting", In Artificial Intelligence and Statistics 2001, pages105–112. Morgan Kaufmann, 2001.

[150] E. Parzen, "On estimation of a probability density function and mode". Annals of Mathematical Statistics, 33:1065–1076. 1962.

[151] J. P. Patist, "Optimal window change detection" in Proc. 7th IEEE Int. Conf. Data Min-ing Workshops, pp. 557–562. 2007.

[152] B. Pérez-Sánchez, O. Fontenla-Romero,B. Guijarro-Berdiñas and D. Martínez-Rego. "An online learning algorithm for adaptable topologies of neural networks". Expert Syst Appl 40:7294–7304, 2013.

[153] B. Pérez-Sánchez, O. Fontenla-Romero, and B. Guijarro-Berdiñas. "Self-adaptive topology neural network for online incremental learning". In: Proceedings of the international conference on agents and artificial intelligence (ICAART'14), pp 94–101. 2014

[154] B. Pérez-Sánchez, O. Fontenla-Romero, and B. Guijarro-Berdiñas. "Adaptive neural topology based on Vapnik–Chervonenkis dimension". In: Lecture Notes in Artificial Intelligence, 2015.

[155] B. Pérez-Sánchez, O. Fontenla-Romero, and B. Guijarro-Berdiñas. "A review of adaptive online learning for artificial neural networks". Artificial Intelligence Review, 49(2), pp.281-299. 2018.

[156] R. Polikar, L. Upda, S. Upda, V. Honavar. "Learn++: an incremental learning algorithm for supervised neural networks" SMC 31 (4),497–508, 2001.

[157] W. M. Rand, "Objective criteria for the evaluation of clustering methods," J. of the American Statistical Assoc., vol. 66, no. 336, pp. 846, Dec. 1971.

[158] J.N. Rao and A.J. Scott. "On chi-squared tests for multiway contingency tables with cell proportions estimated from survey data" The Annals of statistics, pp.46-60. 1984.

[159] S. J. Redmond, and C. Heneghan."A method for initialising the K-means clustering algorithm using kd-trees" Pattern recognition letters, 28(8), pp.965-973. 2007.

[160] S. U. Rehman, A. Ashgar, S. Fong, and S. Sarasvady, "DBSCAN: Past, present and future" Proc. 5th Int. Conf. Appl. of Digital Information and Web Technologies (ICADIWT), pp. 232–238, 2014.

[161] J. Ren and R. Ma "Density-based data streams clustering over sliding windows". In Proc. the 6th Int. Conf. Fuzzy systems and Knowledge Discovery, pp.248-252, 2011.

[162] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," ACM SIGGRAPH Computer Graphics, vol. 21, no. 4, pp. 25–34, Aug. 1987.

[163] G. Roffo, S. Melzi and M. Cristani, "Infinite feature selection", In: Proceedings of the 2015 IEEE International Conference on Computer Vision, pp. 4202-4210, 2015.

[164] P.J. Rousseeuw and L. Kaufman. "Finding Groups in Data," Wiley Online Library, 1990.

[165] E.V. Ruiz, "An algorithm for finding nearest neighbours in (approximately) constant average time". Pattern Recognition Letters, 4(3), pp.145-157. 1986.

[166] C. Ruiz, E. Menasalvas, M. Spiliopoulou. "C-DenStream: Using domain knowledge on a data stream" In Proc. the 12th International Conference on Discovery Science,pp.287-301. 2009.

[167] T. A. Runkler, "Ant colony optimization of clustering models" Int. J. of Intell. Syst., vol. 20, no. 12, pp. 1233–1251, 2005.

[168] S.R. Safavian and D. Landgrebe." A survey of decision tree classifier methodology" IEEE transactions on systems, man, and cybernetics, 21(3), pp.660-674. 1991

[169] A. Saffari, C. Leistner, J. Santner, M. Godec, H. Bischof, "On-line random-forests", in: ICCV Workshops IEEE 12th International Conference on,2009.

[170] G. Sanguinetti, J. Laidler, and N.D. Lawrence. "Automatic determination of the number of clusters using spectral algorithms" In Machine Learning for Signal Processing, 2005 IEEE Workshop on (pp. 55-60). 2005

[171] B. Scholkopf, R. C. Williamson, A. J. Smola, J. S. Taylor, and J.C. Platt. "Support vector method for novelty detection" In Neural Information Processing Systems, pages 582–588, 2000.

[172] M. Scholz , R. Klinkenberg. "An ensemble classifier for drifting concepts" in: Proceedings of the Second International Workshop on Knowledge Discovery from Data Streams(IWKDDS'05),pp. 53–64. 2005.

[173] G. Sheikholeslami, S. Chatterjee and A. Zhang. "Wavecluster: A multi-resolution clustering approach for very large spatial databases". In VLDB (Vol. 98, pp. 428-439). 1998.

[174] P. S. Shelokar, V. K. Jayaraman, and B. D. Kulkarni, "An ant colony approach for clustering" Analytica Chimica Acta, vol. 509, no. 2, pp. 187–195, May 2004.

[175] F. Shen, O. Hasegawa "An incremental network for on-line unsupervised classification and topology learning" Neural Networks, 19 (1), pp. 90-106, 2006

[176] R.R. Sillito and R.B. Fisher, "Incremental one-class learning with bounded computational complexity". In International Conference on Artificial Neural Networks (pp. 58-67). Springer, Berlin, Heidelberg. 2007

[177] G. Song, Y. Li, C. Li, J. Chen and Y. Ye, "Mining textual stream with partial labeled instances using ensemble framework", International Journal of Database Theory and Application, vol. 7, no. 4,p. 47-58. 2014

[178] V.M. Souza, D.F. Silva, J. Gama and E. Gustavo. "Data stream classification guided by clustering on nonstationary environments and extreme verification latency" SIAM-ICDM, pp. 873-881. 2015

[179] K.O. Stanley, "Learning Concept Drift With a Committee of Decision Trees", Technical Report, Department of Computer Sciences, University of Texas at Austin, UT-AI-TR-03-302. 2003.

[180] W. Street and Y. Kim. "A streaming ensemble algorithm (SEA) for large-scale classification" in: Proceedings of the Seventh ACM International Conference on Knowledge Discovery and Data Mining (KDD'01), ACM Press, New York,pp. 377–382. 2001.

[181] N.A. Syed, , H. Liu and K.K. Sung. "Handling concept drifts in incremental learning with support vector machines" In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 317-321). ACM. 1999.

[182] M. J. Tax and R.P.W. Duin. "Support vector domain description".Pattern Recognition Letters,20:1191–1199, 1999.

[183] D.M.J. Tax, "One-class classification: concept-learning in the absence of counter-examples", [Ph. D. thesis]. Delft University of Technology. 2001.

[184] D.M. Tax and P. Laskov, "Online SVM learning: from classification to data description and back". In Neural Networks for Signal Processing,NNSP'03. IEEE 13th Workshop on (pp. 499-508). 2003.

[185] D.M.J. Tax and K.R. Muller. " Feature extraction for one-class classification". In Proceedings of the ICANN/ICONIP 2003, pages342–349, 2003

[186] V. N. Vapnik. “The Nature of Statistical Learning Theory”, Springer-Verlag, 1995.

[187] A. L. Vizine, L. De Castro, and R. Gudwin, “Text document classification using swarm intelligence” Proc. of KIMAS, 2005.

[188] A. Wald, “Sequential tests of statistical hypotheses” Ann. Math. Stat., vol. 16, no. 2, pp. 117–186, June 1945.

[189] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang, “Density-based clustering of data streams at multiple resolutions,” ACM Trans. Knowledge Discovery from Data, vol. 3, no. 3, pp. 1–28, Jul. 2009.

[190] W. Wang, J. Yang, J. and R. Muntz. “ STING: A statistical information grid approach to spatial data mining”. In VLDB (Vol. 97, pp. 186-195). 1997.

[191] H. Wang , W. Fan , P.S. Yu , J. Han.“ Mining concept-drifting data streams using ensemble classifiers” in: Proceedings of the Ninth ACM International Conference on Knowledge Discovery and Data Mining (KDD’03), ACM Press, New York,pp. 226–235. 2003

[192] L. Wang, and H. Shen, “Improved Data Streams Classification with Fast Unsupervised Feature Selection”, In: Proceedings of the 17th IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), pp. 221-226, December 2016.

[193] G. Widmer and M. Kubat. ”Effective learning in dynamic environments by explicit context tracking.” In European Conference on Machine Learning, pp. 227-243. Springer, Berlin, Heidelberg, 1993.

[194] P.D. Wasserman. “Advanced methods in neural computing” John Wiley & Sons, Inc. 1993.

[195] T. Wilk ,M. Woźniak. “Soft computing methods applied to combination of one-class classifiers”.Neurocomputing 75:185–193. 2013.

[196] F. Wilcoxon,and R. A. Wilcox. “Some rapid approximate statistical procedures”. Lederle Laboratories, 1964.

[197] M. Wozniak , A. Kasprzak, and P. Cal , ”Application of combined classifiers to data stream classification” in: Proceedings of the 10th International Conference on Flexible Query Answering Systems FQAS 2013, in: LNCS, Springer-Verlag, Berlin, Heidelberg, 2013.

[198] Z. Xiong, et al. “Multi-density dbscan algorithm based on density levels partitioning.” Journal of Information and Computational Science, vol. 9, no. 10, pp. 2739–2749, 2012.

[199] Z. Xu, K. Yu, V. Tresp, X. Xu and J. Wang.“ Representative sampling for text classification using support vector machines”. In Advances in Information Retrieval, volume2633 ofLNCS. 2003.

[200] B. Xu, F. Shen, and J. Zhao, “A density-based competitive data stream clustering network with self-adaptive distance metric” Neural Networks. 2018.

[201] X.S. Yang.”Firefly algorithms for multimodal optimization” In International symposium on stochastic algorithms (pp. 169-178). Springer, Berlin, Heidelberg. 2009.

[202] Y. Ye , S. Squartini, and F. Piazza, “Online sequential extreme learning machine in nonstationary environments” Neurocomputing, vol. 116, pp. 94–101, 2013

[203] A. Ypma and R.P.W. Duin. “Support objects for domain approximation”. InICANN’98, Skovde (Sweden), September, 1998.

[204] T. Zhang, R. Ramakrishnan, and M. Livny, “Birch: A new data clustering algorithm and its applications,” Data Mining and Knowledge Discovery, vol. 1, no. 2, pp. 141–182, 1997.

[205] P.Zhang , X. Zhu , J. Tan , L. Guo , Classifier and cluster ensembles for mining concept drifting data streams, in: Proceedings ofthe IEEE International Conference on Data Mining, 2010, pp. 1175–1180. 2010.

[206] A. Zhou,F. Cao, Y. Yan, C. Sha, and X. He. “Distributed data stream clustering: A fast EM-based approach”. In Data Engineering, 2007. ICDE . IEEE 23rd International Conference on (pp. 736-745). IEEE. 2007

[207] A. Zhou, F. Cao, W. Qian and C. Jin. “Tracking clusters in evolving data streams over sliding windows”.Knowledge and Information Systems 15,2, 181–214.2008.

[208] X. Zhu, P. Zhang, X. Lin, and Y. Shi, “Active learning from datastreams” inProc. 7th IEEE Int. Conf. Data Mining,pp. 757–762. 2007

[209] I. Zliobaite, A. Bifet, B. Pfahringer and G. Holmes. "Active Learning With Drifting Streaming Data", IEEE Transactions on Neural Networks and Learning Systems, 25(1), 2014.